The Pragmatic Programmers

TURING



Manage It!

Your Guide to Modern, Pragmatic Project Management

项目管理修炼之道

[美] Johanna Rothman 著
郑柯 译
胡百师 审校





人民邮电出版社
POSTS & TELECOM PRESS



Johanna Rothman

世界知名的管理顾问，擅长高科技产品开发管理，经验丰富。1994年创办Rothman咨询公司。她也是活跃的技术作家，长期担任*Fast Company*和*Software Development*等业界著名杂志以及Computerworld.com和StickyMinds.com等一流在线媒体的专栏作者。除本书曾获大奖之外，她的*Behind Closed Doors*和*Hiring the Best Knowledge Workers, Techies & Nerds*等著作也都广受好评。她的博客地址是http://www.jrothman.com/blogs.html。



郑柯

InfoQ中文站敏捷社区（http://www.infoq.com/cn/agile/）首席编辑，《程序员》杂志前副主编。迄今为止翻译了项目管理与敏捷相关的图书、文章、新闻近100万字。与人合译《高效程序员的45个习惯——敏捷开发修炼之道》（人民邮电出版社）。



图灵程序设计丛书

项目管理修炼之道

Manage It!

Your Guide to Modern, Pragmatic Project Management

[美] Johanna Rothman 著

郑 柯 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

项目管理修炼之道 /（美）罗斯曼（Rothman, J.）著；郑柯译．—北京：人民邮电出版社，2009.10
书名原文：Manage It!：Your Guide to Modern, Pragmatic Project Management
ISBN 978-7-115-21361-7

Ⅰ. 项… Ⅱ. ①罗…②郑… Ⅲ. 软件开发－项目管理 Ⅳ. TP311.52

中国版本图书馆CIP数据核字（2009）第157653号

内 容 提 要

本书基于作者多年项目管理的实践经验，融会贯通地讲解了成功管理软件项目的各个要素。书中内容涉及软件项目管理的整个流程：项目启动、项目章程、项目计划、项目日程安排、项目估算、明确的角色和职责、明确的开发流程、恰到好处的度量标准、发布条件、参与 beta 测试的客户……所有成功项目管理的必备元素一应俱全。贯穿全书的提示和生动的案例，更能加深读者对项目管理的领悟。

本书是一本项目经理的实战手册，项目开发人员、软件经理等项目相关人员也能从中获得有益的指导。

项目管理修炼之道

◆ 著 [美] Johanna Rothman
译 郑 柯
审 校 胡百师
责任编辑 傅志红

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：800×1000 1/16
印张：17.5
字数：414千字 2009年10月第1版
印数：1－3 000册 2009年10月北京第1次印刷

著作权合同登记号 图字：01-2008-4301号

ISBN 978-7-115-21361-7

定价：49.00元

读者服务热线：(010)51095186 印装质量热线：(010)67129223
反盗版热线：(010)67171154

版 权 声 明

读者对本书的赞誉

我是一个有三十多年项目管理经验的老兵，这些年的经历让我对有些东西有了深入的理解。在本书中，抽象的流程退居次席，让位给具体的环境和背景。在我看来，约翰娜提供了关于项目管理方面最周到细致的观察、建议和忠告。

► 麦克·德怀尔，Healthways战略合作部门资深经理

约翰娜在本书中收集了很多宝贵的实用建议。即使是最有经验的项目经理，也能从书中找到大量的金玉之言，并且能马上运用到自己的项目管理工作中。

► 詹姆斯·A.沃德，James A. Ward联合有限公司资深项目管理咨询师

我在读这本书的时候，脑海中经常出现许多类似的场景。我有时会想："如果事情是另外一种情况呢？"我却总能发现书中马上谈到了相关的主意！这是我读到过的最棒的IT项目管理图书，而且其中仍然体现了约翰娜的个人魅力。当你阅读的时候，似乎能感受到她就站在你的身边。

► 埃里克·皮特森，Emprove公司资深咨询师

我读过的很多项目管理方面的书籍和资料，大都显得过于注重理论，而有些书却过于具体和武断，与我所关心的问题相去甚远。这本书正好提供了我需要的东西——应对现实状况的有针对性的建议；而且，它还告诉我们如何思考问题，而不仅仅限于给出按部就班的解决步骤。

► 彼得·哈里斯，Claricode公司解决方案架构师

这是一本读起来让人高兴的书，其中充满了智慧。刚入行的项目经理会从中得到一些非常有价值、非常实用的实践介绍，而资深的项目经理则可以了解一些新的工作技巧，同时回顾一些已被遗忘的基本知识。项目的出资人和客户也应该看一看。我的藏书中有一些非常经典，如迪马可、温伯格、布鲁克斯、麦康乃尔、考克伯恩、麦卡锡还有汉弗莱写的书。而约翰娜这本书的可读性堪比其中最佳的作品。

► 乔治·霍桑，Oblomov咨询公司项目经理

近二十年来，我几乎一直被糟糕的项目管理所困扰，始终厌倦于做一个项目经理。可看到这本书之后，我不再这么想了。约翰娜结合实际剖析了项目管理的艺术，整理出了一套实用而灵活的项目管理方法。这套方法建立在多变的环境蓬勃发展的经验性的过程控制理论之上。项目要想成功，坚持不懈、持续学习是必不可少的。我向周围所有参与项目管理工作的人推荐了这本书，并且建议他们多读几遍。

▶ 比尔·克莱伯，宇航工程师

在二十年的项目管理生涯中，项目经理要考虑的新东西层出不穷。约翰娜·罗斯曼的书介绍了许多这样的内容。只看关于会议的那一章就已经值得买整本书了。好好读读这本书，试试实践其中的原则，你的项目团队一定会认定你是个名副其实的聪明人。

▶ 德韦恩·菲利普，高级系统工程师

每个项目都是独一无二的，因此所有的项目经理都得多掌握几种管理项目的方式。约翰娜给我们展示了她的思考过程，从项目背景的评估，到生命周期的选择，直到为项目建立清晰的条件。她的建议能够帮你作出明智的决策，使项目走向成功。

▶ 艾丝特·德尔碧，艾丝特·德尔碧联合有限公司总裁

项目管理实践的真功夫
（推荐序）

我第一次被任命为项目经理的时候，是抱着一颗忐忑的心进入项目的。对项目管理工作的茫然、不安以及种种工作上产生的误区，全都在跌跌撞撞中慢慢地积聚。在二十多年的软件工程生涯中，我一直期望着有一本项目管理的实战图书，真正能够以务实的方式指导项目经理如何面对那些令人头疼又难解的问题。终于，我看到了：*Manage It*——约翰娜的杰作。

一年半前，我在CSDN的SD2.0大会上曾经说过："项目经理这活不是人干的。"当时，这个言论在研讨会场引起一阵回响，会后大家还继续在网上广泛讨论着这个议题。看完这本书之后，我还是有这样的感觉：一个称职的项目经理还真不是一般人能干的。基本上，项目管理是一件十项全能运动，项目经理必须十八般武艺样样精通，尤其是必须敏感地意识到项目的风险。风险管理是项目管理的重心，这一点是我多年项目管理工作的心得。在每一次项目管理的课程中我也会这样殷切地告诉学员们。本书多次提到了应对风险的方法与手段。作为一名软件工程及项目管理的老兵，我深深地认为这本书是项目管理领域中最佳的实践书，尤其是本书作者在项目管理上积累了多年的丰富经验，肯定能为项目经理解答相当多的疑难问题。有句话说：好东西要跟好朋友分享。在此我诚挚地向大家推荐这本好书，分享我在其中得到的共鸣与喜悦。

基本上，每一个项目都存在其独特性，在项目进展的生命周期里，项目管理的行动从售前、投标、议价、组建项目团队、提交项目计划、开展项目工作、进行测试、培训与交付的一连串活动中，就已逐步展开了，这些活动的目的都是为了项目的最终产品交付。项目经理在这个过程中将面对大量的挑战，透过种种的管理行为解决其中的问题。也因为每个项目的特性不同，项目经理接受的考验也不同。项目经理是在软件发展过程中折损率最高的角色，从项目管理相关的教科书与PMP的培训教材里能够学到的是一个项目经理需要具备的基本技能。这些技能只能让你成为一名项目经理，但是无法让你成为优秀的项目经理，其中的差异就在于是否有实战经验的积累。这本书中介绍了很多实用的方法，同时指导项目经理如何灵活交互运用。

多年来，我一直认为项目管理是一种"艺术"，是一种执行管理技巧的"艺术"。这种艺术架构在项目管理的基本技能之上，必须在充分了解项目管理的真谛之后，经由一场又一场的实战，

练就属于自己的管理心法，然后才能随心所欲地发挥。这一层境界是我一直追求的，但至今尚未突破。不过，我在这本书中发现了新的思维，个人的管理功力又增加了一层。我只能说，这真的是一本值得一读再读的好书。尤其在国内广大的IT市场中，项目管理人才十分缺乏，好的、全能的项目经理更是少之又少，我想这本书无论是对高阶管理者或是有心在项目管理领域发展的人，都能让你细细地体验项目管理的艺术。

图书的翻译过程是艰辛的。我有幸看到本书译者对内容逐字逐句的推敲，并与我进行相关的讨论，对于不明确的语意，译者甚至发邮件咨询原作者，努力保存其原汁原味，这种敬业精神让人感佩。如何把一些英文的惯用语，传神地翻译成中文并且让读者能够感受到原作者想表达的境界，这的确是一件高难度的学问。本书的译者做到了。期望广大的读者们在阅读的过程中，能够好好地品味译者的用心与个中的滋味。

胡百师

微软2009项目管理MVP

致中国读者

动手为这本书的中文版写序，我又荣幸又兴奋。感到荣幸，是因为我对中国文化极为崇敬，特别是中国人勤劳工作形成的文化。中国正处在高速发展之中，高效的项目管理也是这种成长的一部分。

本书能够译成中文，我十分兴奋。当我在写这本书的时候，我发现书中的理念可以在西方文化与中东文化情境中畅行无阻，但不太确认东方文化能否接受。现在，我经过长期在世界各地提供咨询、主持研讨会、举办讲座，终于相信本书中的思想可以超越文化障碍，引起中国读者的共鸣。

项目经理可以发现，这本书能够为你节省时间和金钱。对负责项目技术部分的技术领导来说，这本书能够帮你组织并完成技术相关的项目工作。如果你是高层管理人员，也可以利用书中的思想，帮助你的项目经理们考虑更多可行的选择。

我希望你们喜欢这本书。谢谢。

约翰娜·罗斯曼

从自发到自觉
（译者序）

说起来，我也曾作为一名项目经理，带队完成过一个令人难忘的项目。当时，我和另外两名团队成员从北京到南方某大城市的客户现场进行项目开发，负责一个大型电子商务平台系统的定制和运行维护。这个系统有50多个功能模块，其中需要定制或修改的模块将近一半。项目分为多个明确的时间段，并对各时间点有着非常严格的要求，用“时间紧，任务重”来形容这个项目毫不夸张。更让人紧张的是，项目完成到一半时，其中一位团队成员因水土不服，不得不提前返回。不过幸好，经过几方通力合作，虽然有一点点瑕疵，这个项目最终还算是顺利地完成了。

事后分析，这个项目能够结案，运气是主要因素。我当时虽然挂着项目经理的头衔，却并不具有对项目整个图景清晰、理智的分析和认知，没有详细考量项目的驱动因素、约束、风险等。总而言之，就是没有一套成体系的理论、方法和实践。很多时候，支持我作出判断的仅仅是经验和直觉。我相信，有过类似经历和体会的“项目经理”肯定不在少数。

国内项目经理的现况是责大权小。一个项目，自始至终，项目经理都要为需求乱变而担心，为领导改主意而闹心，为团队成员不高兴而操心，为回款困难而累心，有时候还要怕家里人嫌自己加班多而烦心。凡此种种，不一而足。然而，项目经理这个工作也有其独特的魅力。项目实施过程中与团队一起胼手胝足结下的兄弟情义，成功结案后来自客户由衷的赞扬与认可，以及自己心中油然而生的成就感与自豪感，这些都足以让项目经理在放过几天大假之后，再次振奋精神，迎接新的战役。

一个项目就是一场战役，所不同的是：项目战役中的敌人不是实实在在的人，其中各种显而易见或深藏不露的风险才是真正的敌人。很多时候，我们在明处，风险在暗处，而且不同项目千差万别，以往的个人经验并不足以发现某些全新项目中危害极大的潜在风险，这同样要求我们要自觉运用成体系的思考和分析能力。《孙子兵法》、《战争论》等军事经典著作让众多带兵打仗的人有据可依、有章可循。面对战场上瞬息万变的种种情态，他们可以运用书中总结提炼出的精髓和理论，再结合以往的实战经验，以不变应万变，就能做到攻无不克战无不胜。

你手中这本书完全可以作为项目经理的实战手册。大概翻一下就可以知道，作者约翰娜 • 罗

丝曼完全没有老学究般的陈腔滥调。她早年曾与软件业界公认的泰斗级人物、思想家杰拉尔德·温伯格（Gerald M. Weinberg，但人们习惯称他为Jerry Weinberg）共事，并从他那里汲取到诸多软件工程与项目管理方面的经验。此后，约翰娜一直从事与项目管理咨询相关的工作，本书正是她数十年经验的萃取和提炼。我当初做项目经理时，手里要是能有这样一本书，那就可以做到胸有成竹了。

这本书能够翻译完成要感谢很多人。首先是我的父母，他们给予我很大的理解和支持，好多个周末不能回家陪他们，现在想来心中仍然隐隐愧疚。其次是作者，翻译后期，我多次就翻译中的问题与她沟通，她总是那样积极、热情，而且主动提出来要为中文版作序，真希望能有一天当面与她交流。接下来，要感谢昆山中创软件工程公司副总经理胡百师老师，他在百忙之中欣然答应为我审稿并作推荐序。将近二十年的项目管理经验，让他对本书的翻译提出了诸多中肯而独到的意见。还要感谢图灵公司的编辑傅志红，作为本书的责任编辑，她容忍了我一次又一次的拖稿，而且审稿过程中细致入微，发现了不少隐藏的问题。最后要感谢我的宝宝，感谢她一直以来对我的理解、陪伴、支持和鼓励。

最后，希望每位读完本书的人，将来都能以自己做过的项目为豪！

一个敏捷项目管理爱好者：郑柯

2009年5月

序

大家好，欢迎阅读约翰娜的新书。我在软件行业已经有数十年经验了，目前是位于伯克利的Yahoo！的一名总监。不过，也许你听过数字设备公司（DEC，为互联网早期发展奠定了基石）和它的Alpha系统。那是我曾参与的一个意义重大的项目。

在Alpha系统的交付过程中，我扮演了非常重要的角色。那是一个名垂青史的项目：2000多名工程师遍布世界各地，携手开发同一个系统的不同部分。这需要严谨的规划和项目管理才能成功。我们按照为期四年的时间表，在距离目标日期不到一个月的时间内交付了项目。所以，你大概也能想象得到，我觉得自己是个相当不错的项目经理！不过，我后来才知道什么是真正杰出的项目经理。

1996年5月，我决定离开DEC。听说波士顿地区一家大型软件公司正在招产品团队总监，这正是我所期许的挑战，去领导一个陷入混乱的团队。我当时这么想：太棒了！这才是我想要的工作——循循诱导混乱的团队，帮助他们交付可以实际工作的产品——赶紧把我的大洋马牵过来！

我听说有个咨询顾问已经先期加入，试图根据团队产品beta版本的开发状况，分清轻重缓急，帮团队解决问题。这却更加让我坚信：不久之后，他们就会发现，我——才是他们一直在等待的大救星。

可是，我很快就感到了羞愧和震撼（而且感觉越来越强烈）。我知道咨询师们是干什么的，可是他们有谁能够通过实际行动、以实用的方式来厘清所面对的问题？可这位咨询师就做到了。仅两、三个月的时间，她就让一切各就各位：项目章程、工程计划、项目计划、明确的角色和职责、明确的开发流程、恰到好处的度量标准、发布条件、参与beta测试的客户……所有这些成功项目的必备要素一应俱全。

要想把所有这些要素都安排妥当，怎么也得花上大半年时间，尤其是还面临启动资金不足的问题。可事实已经摆在那儿了！你可能已经猜到了，这位咨询师就是约翰娜 • 罗丝曼。（约翰娜在她的网站上放了一个案例研究，就是关于我们这次合作的；为了保护隐私，其中当事人的名字都使用了化名。）

认识约翰娜后的几年里，我先后在大大小小的几家公司里带过软件开发团队。很多时候，我都需要约翰娜的服务，帮助我的团队整体水平再上一个台阶。她的评估流程非常严谨，而且为有效的项目管理活动打下了坚实的基础。她主持的研讨会覆盖了很多话题——给我们讲过的有迭代产生项目需求、项目管理和QA。我曾聘请她出任临时的管理职位，让她使用自己丰富的技能完成一对一的培训指导。约翰娜拥有丰富的经验，曾在很多组织中处理过各种各样的复杂情况，她也总能拿出现实可行的方案，真正解决重大问题。

所以，本书可以说是约翰娜管理才华与丰富经验的结晶。

她把自己多年一线工作积累的经验，以系统严密的方式展现给读者。本书提供了可供你分析所处的实际环境，构建项目管理框架和理性的执行计划，然后予以推进的各种工具。而大量提示和实例，则告诉你哪些路可以走，哪些路行不通，更重要的是如何避开诡秘的陷阱。对我来说，即使已经有了这么多年的项目和工程管理经验，我还是可以在书中发现新东西。当我处于陌生的境况、面临全新挑战时，当我需要一个好参谋帮我应对难题时，约翰娜是我一定要找的人。

我和约翰娜合作的第一个项目最后怎样了？啊对了，我们将产品交付给了做beta测试的客户，而它也确实可以正常工作！

我坚信，约翰娜的这本书同样可以助你一臂之力。

艾伦·R.索尔兹伯里（伯克利Yahoo!研发中心总监）

2007年4月

To Chinese Reader

I'm honored and excited to be writing the preface for the Chinese version of *Manage It! Your Guide to Modern, Pragmatic Project Management*. I'm honored because I have tremendous respect for Chinese culture, especially the work culture. China is growing quickly and part of that growth is effective project management.

I'm also excited about this, the Chinese translation of the book. When I wrote the book, I'd seen the ideas work in Western and Middle Eastern cultures. I wasn't so sure about Eastern cultures. And now that I've been consulting, leading workshops, and speaking all over the world, I am sure that the ideas in this book transcend culture, and speak directly to people.

If you are a project manager, you'll find this book will save time and money. If you're a technical lead who's responsible for making a piece of the project work, this book will help you know how to organize your work and accomplish it. And if you're a senior manager, you can use the ideas here to help your project managers consider other options.

I hope you enjoy the book. Xie xie.

Johanna Rothman

前　　言

说到项目管理，你一定被不计其数的技巧、实践轰炸得头昏脑胀，时不时地还有各种相关建议迎面袭来。它们全都在说："瞧我，我是最正确的。"

哎，它们很多都是正确的——在特定情况下。每个项目都是独一无二的，你必须要评估项目的情境（项目、团队、所在公司），然后再实事求是地作出判断，看看哪些可行，哪些不可行。

你的项目每天都在加快节奏，你的客户变得越来越不耐烦，大家越来越不能容忍无法正常工作的产品。也许你之前的做法还算不错，让你获得不少好评，可将来很可能不太奏效。你必须运用各种方法和技巧来减少项目的风险，这其中就包括在每个项目中使用敏捷方法。

本书从风险角度出发帮助读者规划和指导项目。项目经理、团队成员、软件经理都能通过本书学习成功之道。即使你要构建有形的产品，比如一座房子、某种电路板，或是要管理服务类型的项目，本书中的很多内容仍然适用。

本书假设读者负责管理高科技项目，而且项目至少涉及一些软件开发。也许你像我一样，已经拥有了一些项目管理经验：包括纯粹的软件项目以及软、硬件结合的项目。我也管理过一些服务项目，比如规划和主办会议；参与过一些建筑项目（一套新房子、一次小规模的重新装修、一次大规模的重新装修）。可是我主要的项目经验都来自软件或是软、硬件结合的项目。

相对于交付实体产品的项目来说，软件项目要更加难以管理。软件很难把握，它没有形状，不需要原材料，也不是由物质构成的，所以看不见、摸不着，也没有办法直接测量。很难看到产品实实在在地在我们眼前发生演变，很难发现和预测风险，因此也就更难以应对风险。而开发软件产品的方式也并非总有助于我们了解项目进度或者把握其方向。

如果管理的是开发有形产品的项目，项目经理可以看到产品逐步成型。你可以见到房屋的框架、完工的墙体从框架到墙体，以及所有的建造过程。对于服务型的产品，比如像会议这种会产生具体结果的项目，你可以深入了解一些项目的临时交付物，比如会议报告草稿或是会议日程，等等。所以，在运作项目期间，你可以看到有形产品项目和一些服务项目的具体进度。

要是不能直接看到项目进度，那该怎么办呢？当你发现项目有点儿不对劲儿，而且可能濒临险境时，你该怎么办？如果此时有些项目干系人不支持你的决定，你又该怎么办？

本书可以让你深入了解你的软件项目，并让你成功管理项目的风险，无论这些风险是伴随着项目开始而存在、还是在项目进行到中间阶段时才出现。从章程制定到产品发布，每一章都讨论了一种能帮助你看清软件项目本质的方式，让你从各个方面度量它、感受它、品味它、体会它。

但在本书中，你找不到项目管理的绝对真理，因为没有在所有项目中都颠扑不破的绝对真理。你也看不到普适的最佳实践，我提出的能帮你和你的团队达成目标的实践，都有其针对的特定生命周期。

你在书中会发现有很多前后交叉引用的内容。这是因为项目是非线性系统。早先作出的决策会影响到项目如何结束，甚至可能影响到如何启动下一个项目。你管理项目的方法，也会影响你管理产品待办事项列表或项目组合的思路。

书中的所有文档模板可以在本书的主页上找到：http://pragmaticprogrammer.com/titles/jrpm。

我想感谢所有为我撰写和修改本书提供帮助的人：Tom Ayerst、Jim Bullock、Brian Burke、Piers Cawley、Shanti Chilukuri、Esther Derby、Michael F. Dwyer、Mark Druy、Jenn Greene、Payson Hall、Peter Harris、George Hawthorne、Ron Jeffries、Bil Kleb、Michael Lee、Hal Macomber、Rob McGurrin、Andrew McKinlay、Erik Petersen、Dwayne Phillips、Frederick Ros、Ellen Salisbury、George Stepanek、Andrew Wagner和Jim Ward。我的编辑Daniel Steinberg提供了非比寻常的有益反馈。Kim Wimpsett再次证明他是一个极其出色的文字编辑。我要感谢Steve Peter在排版上的神奇表现。Rotate Graphics的Mark Tatro绘制了日程游戏一章（第6章）中所有的卡通图画。与Andy Hunt和Dave Thomas的再次合作，同样让我深感荣幸。书中任何错误都由我来负责。

书中讲述的故事全部源于真人真事，但考虑到保护隐私，相关的人名、公司名和事件细节都已做过修改。

我们开始吧。

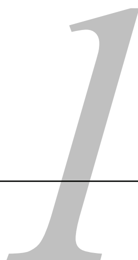
约翰娜 • 罗丝曼

2007年4月

目　　录

第 1 章

启动项目



要想从头搞砸一个项目，最简单的方式就是不动脑子，直接开始。多做一点儿组织和规划的工作，就能为项目多保留一些成功的希望。项目经理必须知道项目的关键驱动因素是什么、项目要怎么样才算"**完成**"，而且还得把这些结论写到章程中，让整个项目团队都能了如指掌。

1.1 定义项目和项目经理

项目到底是什么？我们首先要有一个一致的定义。

> **项目**：一个独特的任务或是系统化的流程，其目的是创建新的产品或服务，产品和服务交付完成标志着项目的结束。项目都有风险，并且受制于有限的资源。[①]

项目经理负责管理风险和资源。交付日期迫在眉睫，技术目标难以达成，产品质量漏洞百出，项目资金即将告罄，人手面临匮乏困境——没有项目管理，贯穿于项目中的这些风险如何应对？

因为各自的关注点不同，每个项目都是独一无二的。项目经理要根据每个项目不同的实际情况，进行管理和规划。在启动之前，着手收集信息，了解项目要实现哪些目标。

我们的目标是什么？

——克里斯，项目经理

> 我是一个大公司的项目经理，主要负责软件和硬件的集成项目。我的同事妮娅负责主办各种活动。我的团队搭建电脑系统，而妮娅的团队负责活动组织——虽然彼此的产出根本不沾边儿，可我们都是项目经理。
>
> 我们面对的风险完全不同，对外交付的东西也根本不一样。我要的是软件、硬件以及一些文档，而妮娅需要展览摊位、食品以及其他有助于活动成功的必备之物。

① © 2007 R. 麦克斯 • 怀德曼，http://www.maxwideman.com。经许可后转载。

我们的工作有一个共同之处：开始时要知道项目的目标，这样就可以马上知道我们接下来要做什么了。知道要做什么、“完成”意味着什么，这对我和妮娅的团队很有帮助。

每个项目都是独一无二的。项目团队规模不同，各自的能力也有所区别。有些项目的客户是内部的，有些则是外部的。有些项目面临很紧迫的时间压力，有些项目则可能要面对其他威胁和风险。产品是每个项目的核心，让我们再就产品的定义达成共识。

产品：项目产生的一系列可交付物。

作为一名成功的项目经理，或者想成为成功项目经理的你，应该花些时间来发现当前项目的独特之处。这样，你就有可能成功地启动、管理和结束这个项目。

我们现在已经有了项目和产品的定义，接下来该搞清楚项目经理的职责了。怀德曼说，一个项目经理“被赋予管理项目的权力和责任，并带领项目团队，通过项目管理来达成项目的目标”。[①]这个正式定义还挺不错的。不妨看看下面这几句大实话，比较而言，也许它给人的感觉更加模糊，但同时也更准确。

项目经理：负责向团队清晰说明完成的含义，并带领团队完成项目的人。完成，是指产品符合组织对这个产品的要求，也能满足客户使用这个产品的需要。

其中，完成的说法是暗含风险的。我可以确定任何项目的产品一定伴随着风险，我们会在后面讨论如何对这些风险进行识别和分类。在采取任何进一步行动之前，项目经理必须理解项目的关键驱动因素是什么。

无论规模大小——是项目就有风险

——埃里克，资深项目经理

我曾参与过延续数年、团队达数百人的大项目；也参与过一些小项目，我们的团队只有两三个人，客户也只有两到三个人配合，整个项目持续时间三个月。关于项目，我有一点非常确定：通往最终交付的道路上，总是有风险伴随在我们左右。

有时，风险因团队具体情况而不同。设想整理床铺这件简单的小事。对我来说，这只不过是待办事项清单上的一个条目而已。可我的孩子们却觉得，铺床就像是一个满是风险的项目，最主要的风险就是他们无法在睡觉之前把床铺好。

1.2 管理项目的关键驱动因素、约束和浮动因素

要完成一个项目，如果不能理解项目的背景，前进的路上必然充满艰险。

① © 2007 R. 麦克斯 • 怀德曼，http://www.maxwideman.com。（经许可后转载）。

项目的背景是由所在组织的关键因素决定的。项目的驱动因素是什么？是否存在约束？有没有可能在驱动因素和约束之间进行折中？项目经理能否为自己争取更多的自由运作空间？

当前项目的驱动因素与上一个不同

——斯图尔特，项目经理

我现在正在管理我个人的第二个项目。第一个项目的交付物是公司旗舰产品的一个补丁版本，因而很容易就能知道我们的工作重点：添加几个功能特性，并修复多个缺陷。

但是这第二个项目——哎，可完全不一样了。这个项目会被用作目标市场的探路石。我们需要很多新功能，这些功能还得没啥大问题。工作重点不是解决缺陷，因为交付日期马上就要到了，得赶紧让产品投入市场。老板跟我说可以多给我几个人，但是不会有外包公司参与，因为要控制成本。

能够识别出哪些事情在推动项目让我获益匪浅。我发现优先级最高的就是功能，其次是交付日期，接下来是人。只要能发布，公司对于缺陷和成本卡得就不是那么死了。

也许大家都听说过项目的“铁三角”：成本和时间是这个三角形的两条边，第三条边一般是质量或范围（见图1-1）。其实铁三角过于简单了。斯图尔特的两个项目的驱动因素就存在很大差异。

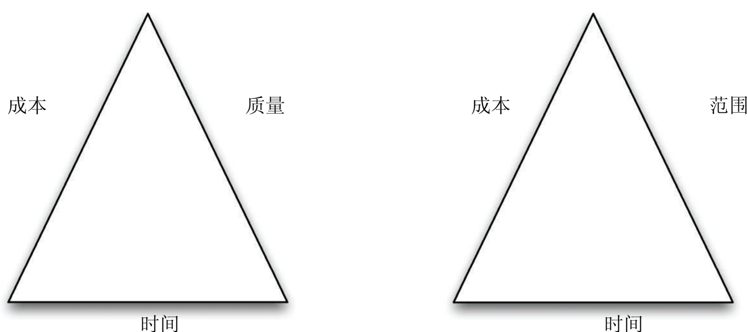


图1-1 传统的铁三角

像斯图尔特这样成功的项目经理会权衡许多因素，而不仅限于铁三角。以为只要让客户从铁三角中选择他最看重的两条边，然后就可以交付他想要的结果，这根本是痴人说梦。要是这么简单的话，那任何人都能当项目经理了。

首先，要写下客户的期望——从客户的角度来看，项目的驱动因素是什么［Rot98］。这个列表中应该包括：客户想要什么（功能集合），他们期望何时收到交付物（发布时间），可交付物的质量如何（缺陷等级）［Gra92］。

接下来，写下项目的约束。环境是什么样子的？能否灵活安排团队的位置？必须遵守什么样的流程？手下的人有哪些？他们能做什么？预算有多少？这些约束是可以改变的（一般只有项目出问题的时候才能改变）。约束决定了项目的规模（还有持续时间和质量）。

对比客户的期望列表和项目的约束列表。你脑海里蹦出来的项目成功的必要因素有哪些？选择其一，比如发布时间，这就是识别出来的项目的**关键驱动因素**。

列表上还剩下什么？应该还能看到功能集合、低缺陷率、发布成本等条目。在这些条目里，需要对哪些进行管理才能保证项目的成功？可以按重要性排列这些关注点。客户和出资人会在这些方面对项目形成限制。从中选择两到三项，我们称之为项目的**约束**。

再看看剩下的条目。有些条目很重要，不过它们有很大的调整余地，我们称之为**浮动因素**。一个项目至少要有三个浮动因素。

最后看看还没选择的条目，是不是还有哪个比已经选择的更重要呢？如果有，那就再调整一下；如果没有，这个项目的关键驱动因素、约束、浮动因素就都识别出来了。

我曾经见过有三个约束的项目，团队必须付出超乎寻常的努力，才能保证项目成功。以我的经验，如果项目有一个关键驱动因素、两个约束、三个浮动因素，那它的成功几率还是不小的。浮动因素越多，项目就越容易管理。

理想状况下，关键驱动因素应该只有一个，约束应该只有一个，而浮动因素可以有四个。大部分情况下，我们管理的都是不那么理想化的项目，所以如果项目有一个关键驱动因素、两个约束、三个浮动因素，这也是可以成功的。过多的关键驱动因素或约束，会让项目过于受限。这种情况下也许能够成功，但是项目经理和团队就得选择促成项目成功的组织形式和实践——不过还是要有点心理准备，因为可能无法获得百分之百的成功。

如果存在过多的关键驱动因素和约束，而且项目经理除了启动项目之外别无选择，这时所能做的就是选定一个关键驱动因素，并尽可能频繁地向出资人提交项目的产出，帮助他们决定自己真正想要的东西。在启动项目的时候，为了得到项目的成功条件，可以问一些与上下文无关的问题（请看1.5节）。还要定义出发布条件（请看2.3节），这样就知道到什么程度算做完了。

提示：要是约束太多了，你就自己作决定吧

受到过多限制的项目难以逃避失败的厄运。过多的关键驱动因素，意味着没有人知道成功的条件是什么。过多的约束，意味着组织中没人愿意去判断孰轻孰重。

有必要的话，让管理层决定项目的关键驱动因素、约束和浮动因素。如果这不可行，那项目经理就自己做决定吧，项目和组织会因此而受益。

想创建出不受约束限制的项目需求是不现实的。假设有一个可以承重5斤的书包，如果你硬要往里面塞10斤的书，不外乎两种结果：要么书包带断掉，要么书包底破掉。管理项目也是如此，要根据约束来管理需求。要想同时应付过多的需求，那不管用什么办法，人手、时间、预算还有工具这些资源可能就是不够用，发布出的产品也很难具备高层想要的全部功能，同时可能有很多缺陷。

1.3　与客户或出资人讨论项目约束

要主动理解出资人想要的东西。下面这段谈话就是我在最近的项目中与出资人的对话。

JR：哎，克莱德。咱们看看到底是什么在驱动这个项目吧。

克莱德：可别！别再整我了。上个项目的时候，你就让我做过一次了啊。

JR：没错。还记得吗？你当时想添加新功能。你希望在产品发布之前塞进去尽可能多的功能，我可以加，因为当时组织项目的方式还允许。那确实挺不好弄的，不过还是有可能做到。

克莱德：啊，对，我忘了。不过这个项目不一样啊。

JR：哦？跟我说说。

克莱德：你瞧，管人是你的事情，项目的运作情况你也得管。不过你不需要资金，我也不担心成本问题，因为唯一的成本就是工资。

JR：还有软件呢。

克莱德：你还真挑啊。好吧，如果需要什么软件，跟我说。不过老实讲，基本上工资就是这次的全部成本——我也不用管，我最关心的是项目要多久才能完成。

JR：那需要哪些功能呢？对它们的质量有什么要求？这是给财务部门用的一个小程序，你也知道他们总是要求完美。要是我不能保证给他们的东西毫无问题，莱斯丽会跟上面吹风的。

克莱德：没错，不过我会帮你扛着的。我就是希望给他们够用的功能就行了，这样你跟你的团队就能很快搞定，比如在十周之内。

JR：如果到了第八周，我们还没有开发完所有的功能，而且还有很多bug要改。你会怎么想？

克莱德：JR，别啊。我需要全部的功能。你们开发团队曾经完成过类似的工作，我相信这次也一定行。

JR：克莱德，如果我能理解你到底想要什么，你知道，我们开发团队会尽力而为的。

克莱德： 好吧。千万不能有不完整的功能。你尽管开始，把它做完，而且要让莱斯丽喜欢它。否则，她非把我头砍下来不可。文斯已经启动了一个大型的项目计划，过段时间，我想让你们开发团队加入到那边去。他说已经准备好在十个星期后接收更多的人了。这样，你也有足够的时间来给莱斯丽一点儿用得上的东西。

1.4 决定项目的关键驱动因素

在与克莱德的对话中，我让他说明他最看重的是什么。克莱德说一个大型计划会在十个星期之后启动，很明显，日期就是这个项目的关键驱动因素。

在项目早期，似乎一切皆有可能，特别是没有估算任何工作的时候。出资人可能会说："我们想要这5个功能，在8月1日之前完成，还不能有严重的问题。我们还想让你把成本控制在一百万美元之下。给你这6个人。OK？"

可别轻易说OK。

估算了工作量之后，就能知道用这么多钱、这么长时间，这6个人能不能做完项目。要是可以，蛮好。不过很可能没有办法按照出资人对时间、金钱、人力的要求，在规定的工作环境中，提供符合他们质量要求的产品。此时，出资人需要作出艰难的决策，要考虑组织中的项目驱动因素，比如发布日期、功能集合、成本、人员分配及其分配时间、将要采用的技术和实践，等等。

要是出资人不愿意判断项目的驱动因素是什么，这个责任就落到项目经理肩上了。如果项目经理不做决定，团队自己会决定。但是他们会各拿各的主意，而且这些主意也不一定会符合出资人的想法。实际上，每个人——不管他或她在项目中的角色是什么——都会根据自己的想法进行判断。有的人会优先考虑发布日期的限制，从而把减少缺陷的想法抛在脑后。有的人会根据日程安排，仅仅实现一个功能——一个包含完整回归测试套件的功能，其余的功能却都没做完。有的人会优先考虑实现功能集合，开发出尽可能多的程序桩（stub），当测试人员发现问题时再去填充，直到时间用完为止。每个人都会做自己认为正确的事，而不是从出资人（或项目经理）的角度出发，因此作出的决策彼此不同。

通过制定优先级列表可以解决这个问题，其过程跟做数独游戏有些类似，即将所有可能的驱动因素列在左边，右边空出来等着填数字。

使用矩阵表明项目的优先级

下面是克莱德的排序矩阵。

项目驱动因素	排序
发布成本	5
发布日期	1
功能集合	2
减少缺陷	3
人员配备	4
工作环境	6

在这个项目中，发布日期是最主要的驱动因素。如果产品今年不能发布，这个项目就没有什么存在的意义了。但是完备的功能也很重要——功能不齐全，即使及时发布，整个项目也没有价值。而且，由于公司业务属于受政府管制的行业，产品的缺陷率必须很低。接下来是人员配备，因为只要这些人能在十个星期之后参与下个项目计划就可以了。项目的成本控制不太重要，因为项目的价值会很高。工作环境排在最后，为了保证及时交付，我可以灵活调整某些事情。

即使已经有了排好顺序的优先级，我们还是没有多少灵活性。不过了解了项目的关键驱动因素，我就可以定义出项目的成功条件，并选择适合项目的生命周期了。项目团队可以制定出发布条件，并根据驱动因素合理地安排各自的工作。嗯，最后，我们按照当初的要求，成功地交付了项目。

1.5 应对喜欢过多干预项目的出资人

在绝大多数情况下，关于成功标准的谈话不会进行得那么顺利。读者也可以看到，我必须促使克莱德作出选择。类似情况很常见。

有些项目对于功能集合、减少缺陷、时间安排这三者的要求都是最高的，而且优先级相同。我敢说，读者一定以项目经理或团队成员的身份参与过类似项目。你不能再给团队添加更多人手了，而且项目的预算不可改变。项目流程必须按照公司的强制要求执行，还得使用公司规定的办公室和家具，其他一些工作环境问题也让项目难以推进。除非没有技术或时间上的风险，否则没人能够使这样的项目取得成功。不过还是有一些方法，能够理清这些看来已经没有活动余地的约束条件。

在1.4节中可以看到项目的驱动因素矩阵。如果出资人愿意排定优先级，那这种方式再好不过了。有些勉强的出资人可能因此而变得更加坚定。项目经理有时要先草拟一个优先级列表，拿给出资人看。比如，几个出资人对于优先级的排序可能无法达成一致意见。要是出资人不愿意拿主意，项目经理就只好自己做决定，然后再给出资人看。她可能会愿意去修正列表，或者直接签

名通过这个列表，自己就不再建新的了。

要想明确项目真正的驱动因素，有两种不同的途径可供选择：预测未来；使用与上下文无关的问题。

1.5.1　预测未来

让出资人设想这样的情况：现在离项目预定交付时间只剩三个星期，却还有功能尚未实现。（可以着重讨论该出资人需要的一两个功能。）除了没做完的功能之外，还有很多严重的缺陷等着修复。这样看来，要想在预定的发布日期之前开发完所有的功能并修复所有的缺陷，很明显是不可能了。这个时候，出资人该怎么办？

如果出资人这么说："把你的脑袋砍下来给我吧。"那就该考虑"三十六计走为上"了。请看7.7节。

不过出资人很可能会这样说："把这个该死的项目做完吧。你还需要几个人？"接着，你可以问他："是先做完所有的功能，还是先修复所有的缺陷？"他一般会说："这两个功能和这些问题都得解决掉。"再问他："以项目目前的优先级顺序，是吗？"项目经理要经常帮助出资人搞清楚：哪些约束是真正的约束，哪些是出资人自己一厢情愿的想法。

在对话中使用与上下文无关的问题，有助于识别出项目的驱动因素、约束以及活动余地。

1.5.2　使用与上下文无关的问题识别项目真正的驱动因素

明确了驱动因素之后，项目经理就可以发掘**项目**的需求了。与上下文无关的问题有助于抽取出这些需求。通过这些比较抽象的问题，可以诱导其他人说出他们对于项目的假设。不妨从下面这些问题开始。

- 项目要怎么样才算成功？
- 为什么想得到这样的结果？
- 这种解决方案对你来说价值何在？
- 这个系统要解决什么样的问题？
- 这个系统可能会造成什么样的问题？

要注意，少用"为什么"之类的问题。"为什么"问得越少，对于业务需要反而能了解得越多（而且问问题的人也不会像个令人厌烦的三岁小孩子。）同时，"为什么"这类问题很容易让对方产生戒心。也得注意避免"怎么做"之类的问题，出资人会觉得你在让他们设计系统。

在问问题时，要让人感觉到你真心希望了解这个项目，而不要让别人抱有戒心。这些问题可以为项目经理和出资人将来的合作打下良好的基础，而不是形成龃龉的关系。

这些问题可以找出系统的价值所在。在向出资人提问时，要营造平等的气氛。在纸上做笔记，而不是用电脑，这样你和出资人之间就不存在障碍了。将这些问题作为谈话的开端。在刚开始时，从出资人处收集到的关于项目价值的信息越多，你就能更深入地了解如何设计这个项目。

项目要怎么样才算成功？

——贾斯汀，项目经理

几个月前，我开始管理这个会持续两三年的项目。这两天，老板跑过来跟我说他想加入一个新的功能。一个我在说：“酷啊，这个新功能一定会很棒的！”而另一个我说：“噢不，我们介入这个项目已经两个月了。要是我给老板留下可以随意加入新功能的印象，如果不调整我们团队的工作方式，那我一定会有麻烦的。”于是，我问老板这个问题：“对你来说，项目要怎么样才算成功？”

使我大吃一惊的是，老板认为这个项目要能完全随需应变才算成功。他十分确定：不到最后一刻，需求不会停止变化——也许要到我们发布前一周才能最后明确。我以前可从来没有管理过类似的项目！不过既然现在知道了，我就能着手筹划应对之策了。

1.6　编写项目章程，共享现有决策

项目章程会明确记录项目的需求和约束，还可以帮助项目经理思考如何进行项目规划。整个团队和出资人都可以查看项目章程，以此确保他们对项目有关的决策可以达成一致。

在启动项目时，编写项目章程可以让大家知道应该完成什么目标，以及项目干系人提出的约束条件。即使不知道完成项目需要的细枝末节，编写章程也有助于发现潜在的问题。项目章程可以帮助项目经理和团队理解风险、成功标准，而大家也可以藉此考虑组织和操控项目的方式。

如果你在管理一个敏捷项目（使用敏捷生命周期的项目，请查看附录A.4节），项目章程会很简短，只要包括项目远景（参与项目的人可以做出正确的决策）和发布日期（这样就不会为了给产品“镀金”而错过项目的结束时间）就够了。

下面是我的项目章程模板。

- 远景
- 需求
- 目标

- 成功标准
- ROI估算

项目章程是有意要设计成这么简短的，目的是帮助团队赶紧启动。它不会包含团队对于这个项目“完成”的定义，也不会介绍团队如何组织项目，但是已经足够让大家着手开展工作了。

1.6.1　远景

每个项目背后总有一个缘由（或者两个、三个）。发起这个项目的缘由何在？项目的价值何在？要用描述远景的句子来说明项目的价值。越早向大家阐明项目的价值，团队就越有可能告诉你接手这个项目是否明智。也许他们会告诉你，囿于目前的约束、资源和他们的能力，这个项目就不可能完成。要是不能明确阐明项目的远景何在，很可能这个项目就是“不可能完成的任务”了（因为没有远景的项目无法结束）。实用的项目远景对团队来说不可或缺。

1.6.2　需求

某些情况下，项目唯一的需求就是要在某个特定日期到达之时发布某些功能。更多时候，需求掺杂在下面这样的语句之中：“2月20日发布的主要版本中，我们需要这个又棒又炫的功能。”这些才是项目的驱动因素，产品功能列表不是。

1.6.3　目标

项目目标是希望通过项目要达成的目的，不过客户跟出资人可能并不赞同这些目标。（想了解更多关于目标的讨论，请查看2.3节）。



小乔爱问……

谁来撰写项目章程呢？

撰写章程，可以帮助团队早点形成凝聚力。要让尽量多的团队成员参与编写章程。

要是还没有为项目分配人员，那就先自己写章程吧。如果不是单枪匹马，就跟大家一起写。在项目启动会议中（请查看10.3节），把章程跟大家过一遍，确保每个人都知道其中的内容及其缘由。

目标与需求不同［DeM97］。项目并不一定必须交付它的目标。遵循传统的阶段–关卡（phase-gate）式开发过程或瀑布式开发过程的项目，它的产品里面会有比较严重的技术债务（technical debt，请查看附录B）。通过重新设计解决技术债务、添加更多的自动化测试、设计冒

烟测试等等，这些都是可能的项目目标。

客户有时会提出一些特定的要求："如果目前的时间安排允许，并且不麻烦的话，我想要一个电子签名。"作为一个注重实效的项目经理，你可以说："OK，我们会把它加到项目目标里面去。要是雪丽和简有时间的话，她们会去做的。"

1.6.4 成功标准

成功标准是围绕客户能基于完成的产品做什么给出的定义。成功的标准并不涉及缺陷，而只关注能力。（请看［Rot02b］和［Wie05］）。

下面是一些成功标准实例。

- 要包括功能1、2、3，这样我们的产品就可以打入目标市场了。
- 要提升产品性能，再测出相关数值，这样我们就可以将其与竞争对手的产品进行对比，并编写新的市场营销材料了。
- 客户不需要访问我们的网站，就可以打开安装包、加载软件。
- 在第一季度发布。

在你意识到之前，项目就已经开始了

项目通常在正式开工之前就已经启动了。也许有人已经开始了原型化的工作，也许有人已经预想了一些功能，也许管理层已经跟首席架构师探讨了项目在技术上的可行性。这些都是项目的组成部分，即使你不这么认为。

由于项目在你思考之前就已经开始了，所以大可不必因项目章程、项目规划和项目日程的反复修改而烦恼。作为一个注重实效的项目经理，在项目前期，虽然有些工作已经开始了，你还是应该张开双臂拥抱项目，接受眼前的一切。在项目中期，你应该不断评估项目进程和风险，从而掌控整个项目。项目收尾阶段，如果你是注重实效的，那就没什么好担心的了。

团队要控制成功标准。项目经理要确保成功标准中不会包含非项目人员才能完成的任务（比如"卖出50 000套软件"）。项目经理和团队不能控制别人做什么，只能控制自己和团队的行动。要确保成功标准在项目经理的掌控之中。

1.6.5 ROI

我接触过的客户中，很少有人会去度量ROI（即投资回报率）。毕竟，要操纵数字太容易了。不过，如果管理层要求的话，项目经理可以估算项目的回报，试着计算ROI。在项目完成后，可

以看看是否达到了当初预计的数字。

1.7　理解质量对于项目的意义

要想理解质量对于出资人和客户的意义，必须先要理解项目的关键驱动因素、约束和浮动因素。出资人是为项目提供资金的人，客户是使用产品的人，他们不一定是同一群人——对质量的定义可能有差别。没错，这让项目经理的工作更不好做了。不过，知道了对于项目来说“质量”意味着什么，也就部分理解了“完成”的含义。

温伯格这样说：“质量，就是对于某人的价值。”［Wei92］这个定义也就意味着可以为项目产品添加新功能，并可查看一个功能吸引（或排斥）了多少个（以及什么样的）“某人”。在想到出资人和客户的时候，不妨考虑一下他们希望从项目中得到什么。这样一来，项目经理便可以根据实际情况调整发布时间、项目预算和人员配备；同时，项目经理也可以权衡：是否所有的“某人”都需要某个功能。如果项目经理和团队知道“某人”对于质量的定义，大家就可以朝着这个方向来努力；即使他们改变了主意，团队仍然可以取得成功。

根据项目产品所处的不同市场应用阶段［Moo91］，如图1-2所示，客户对于质量的定义也有所差别。

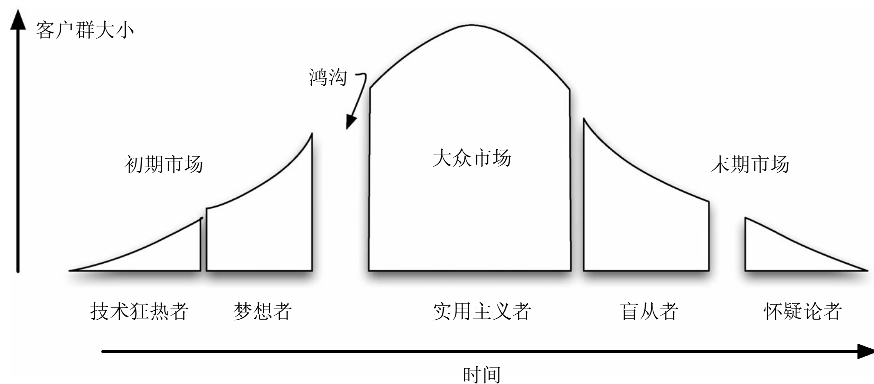


图1-2　摩尔的鸿沟

当产品处于市场应用的早期阶段时，顾客群的规模不大，但是技术狂热者们已经想先睹为快了，所以会面临比较大的发布压力。此时产品不必具有很多功能，而且也不必有很高的稳定性，但是它必须具备自己的独特之处，来吸引更多客户。

在产品进入市场的初期，无论客户群大小，人们都会有各自不同的需求。所有客户都希望马

上得到新版本，而且要具备他们自己想要的功能。此时的产品用起来不能太费事，不过由于只有你的产品能够解决客户的问题，所以即使有点儿缺陷，它的销路还是会很不错。

一旦产品（或有替代产品）进入到大众市场之后，实用主义者们就会关心产品的缺陷能否得到及时修复。如果在之前的版本中累积了技术债务（请查看附录B），现在就到了还账的时候了。鉴于实用主义者拥有强大的购买力，管理层会迫于压力而决定频繁升级产品——即使有些客户不想安装最新的版本。在加强产品稳定性的同时，每次发布的新版本中多少要加入一些新功能。

保守主义者也会购买你的产品，但往往是迫于某种压力作出的选择。如果产品不能完成承诺的功能，或者有太多缺陷，保守主义者们会抓住一切机会发起抨击。他们不需要新功能，只希望承诺的功能好用就行了。此时只要发布缺陷修复补丁，或是发布更加可靠、更高性能、更稳定的产品即可。

对于你的产品，盲从者和怀疑论者有可能买，也有可能不买。处于这个阶段的产品有时会被称为“现金牛”，因为此时产品为公司带来的收入要远远超过需要付出的成本。

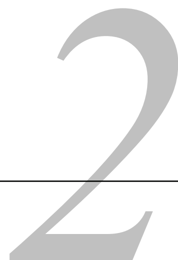
尽管在产品生命周期的开始阶段发布的很多版本都会面临不小的压力，但在产品生命周期的后续阶级，规模日益增大的客户群对低缺陷率要求的压力会更大。

铭记在心

- 每个项目启动时都要有章程。
- 对项目章程的反复修改要有心理准备。章程不一定完美，它的意义在于帮助整个项目团队进行规划活动。
- 要知道“质量”的意义以及项目的驱动因素。这样随着项目的不断推进，项目经理和团队才可以作出正确的决策。

第 2 章

规划项目



现在你手里有项目章程了。要是团队成员不能跟你一起编写章程，那就在项目启动会议上跟团队成员一起把它过一遍（请看10.3节）。要是团队成员已经熟悉了章程，那就可以跟他们一起做些有目的的规划和日程安排工作了。

规划和日程安排是两种不同的活动。规划是指制订带有发布条件的项目计划，而日程安排则是对工作项目的有序描述。

2.1 踏上征程

有了项目章程，每个团队成员就可以对自己接下来要干什么做些有明确方向的预先规划，或者，也可能提早知道自己还没有明确的方面。有了项目规划，就可以把团队成员的注意力聚集到预期的项目产出上来。

如果团队的人不多，比如不超过十个人，不妨与他们一起完成项目规划。跟团队成员商量接下来的几天或几周之内要做什么，同时要将项目的产出牢记心间。如果已经收集到一些需求，团队就可以开始原型化的工作，或是展开第一次迭代的开发工作了。要是采取敏捷开发过程，不妨将项目规划看作发布计划。

不过事情不会这么完美，比如人手不够，或者有太多的人参与原型化，甚至你根本不知道接下来该干什么。此时，可以让一些人或全部团队成员去修复上个版本留下的缺陷。请看13.1节。

2.2 使项目足以启动的规划

章程有了，规划是什么？管理层希望知道团队什么时候开发哪些特性。如何测量进度？项目何时完成？

规划不必完美无缺。实际上这也是做不到的。只要这个规划能让项目启动起来，同时能让大家看到成功的希望，就可以了。如果项目面临时间的压力（我接手的项目大多如此），那就要用时间盒（timebox）来辅助规划活动。即使打算重新规划某个项目，项目经理也不必在一开始担心规划是否完美。

时间盒是指特定的时间长度，个人或团队用它来完成某项特定的任务。个人或团队在这段时间内完成的工作量，就是项目接下来的工作的基础。如果有必要，个人或团队可以减少工作范围，以保证在“时间盒”内完成工作。

用时间盒来限制启动规划活动

——史蒂夫，项目组合管理高级总监

一般来说，我们会同时进行2～5个大项目以及15个左右的相关小项目。对于绝大部分项目来说，我们在启动时会先预计到底需要多少人力以及将会耗费多少时间。即使是为一个新项目做初始规划，也要保证耗时不长。

我们最近的一个大项目，预计耗时两年左右，需经历多次发布。我们把制订初始规划的时间盒设定为三天。三天过后，我们有了项目章程、项目规划，并制订出了第一次发布的发布条件，以及前三周的工作日程安排。就是这些，不过已经足够启动项目了。

我们最近的几个小项目（耗时不到半年），制订初始规划的时间盒设定为一天，其中包括安排一个为期两周的工作日程。当项目工作展开后，再根据执行初始规划得到的反馈，指导后续的规划。

有些项目已经完成了某些特定的功能，需要预测这些项目的发展和走向。我们会在项目前几周内进行初步的预测，后续几周内更新这些预测。到了8~12周的时候，对于何时完成哪些功能，我们心里也就有底了。我们没有做到尽善尽美，但是也不会在没有数据的情况下，浪费时间做过多无效规划。做一点规划，收集一些数据，再重新规划。这样做无往而不胜。

即使项目没有面临时间压力，要是为了得到“完美”的规划而花费太多工夫，时间的要求也会使这个项目变得越来越紧迫。

提示：要根据经验而不是预言来规划项目

经验式规划也就是指不妨先做少量规划，再根据实际过程中收集到的信息反馈来影响未来的规划，这样做具有很高的可行性。预言式规划则是试图预测未来，是很难奏效的，除非你有水晶球。应该尽量在项目中使用经验式规划和日程安排。

也许你听过艾森豪威尔那句名言，“规划毫无用处，但是制订规划必不可少。”[①]项目有太多的风险和不确定性，想在一开始就把一切都想好势比登天。从规划开始，每隔几周再重新规划，就像5.6节所述。无论采取何种生命周期管理项目，都得做好反复规划的准备。不要创建完美的规划，只要这个规划在被（很快）替换之前具备可行性就可以了。

提示：以终为始 [Cov91]

开始规划时，要想清楚有多少内容需要写下来。发布条件（请看2.4节）可以使出资人和项目团队集中注意力。如果想减少制订规划的时间，至少要保证规划出发布条件和风险列表。必要时要反复修订规划。如果使用敏捷生命周期来管理项目，就不要做任何预先规划活动了。[Coh06]

2.3　开发项目规划模板

项目经理要用项目规划来整理想法，最好选择能在组织的更多项目中重用的形式。下面是我的项目规划模板：

- 产品意图
- 历史记录
- 发布条件
- 目标
- 项目组织
- 日程总览
- 人员配备（人员曲线）
- 建议日程
- 风险列表

2.3.1　产品意图

简单描述产品，为什么公司要开发这个产品，它能为公司带来哪些效益，等等。发布版本的价值何在，它是否是后续发布版本（用三四句话）？如果已经在章程中写下了项目的远景，不妨在这里借用一下。

---

① 1957年11月14日，艾森豪威尔在华盛顿特区国防行政准备会议（National Defense Executive Reserve Conference）上的讲话。

有时候，章程中的远景对于项目来说不够用。或者说，远景是供大的项目工程（一系列或者一组项目，请看14.1节）使用的。这类情况下，要确保当前项目的意图清晰无误。

多年以前，在TCP/IP协议还没有被整合入操作系统之前，我管理了一个工程，其产出是一个软硬件结合产品，其中包含了许多特性，每个特性都是一个单独的项目。我为工程制定了章程，其中的远景描述如下："及时发布产品，供某某商业展使用。"有六个项目团队，网络团队是其中一个。他们对项目的意图定义是："确保TCP/IP在第一次集成之前就能正常运作，并在整个项目中一直发挥作用。"他们的意图很清晰，而且与工程的远景相关，同时又非常明确。

基于项目（或工程）的规模和持续时间，每个项目团队（或子项目团队）都有其自己的意图。这个意图应与工程章程的远景在大方向上保持一致，但又不完全相同。

2.3.2 历史记录

如果是在管理某产品的后续版本，比如4.2版本后的4.3版本，就要复查之前或相关版本的历史记录。这个历史记录可以说明之前任何已知的技术债务（见附录B）。要复查的数据包括：发布频率、发布后发现的缺陷数、客户报告的问题，以及会影响到项目经理对质量的定义、对项目管理方式的任何内容。

提示：知道的越少……

对项目情况知道得越少，感到惊讶的几率也就越大。这适用于技术债务、新的架构、开发或测试风险或者制订规划等多个方面。只要在之前的项目中没有积累类似经验，就很容易遇到意外的情况。做一些规划，并做好反复规划的准备，这可以助你一臂之力。

如果客户习惯于每年发布一次产品，并且不会提出更多要求（按照摩尔的说法，他们已经是"大众客户"了，请看图1-2），此时项目经理有多种选择。第一种是拒绝管理层对频繁发布的要求。除此之外，可以使用版本火车（请看14.3节）或敏捷生命周期来进行更频繁的发布。如果需要打开或进入新的市场，在组织项目时，要考虑使用适合产品频繁发布的方式。

复查已发布版本中发现的缺陷的数量和类型，就可以理解即将使用的代码库中技术债务（请看附录B）的严重程度和类型。

如果客户在某个领域中报告了大量的问题，要看看到底是文档还是代码的问题（或是其他类型的问题）。对即将遇到的技术债务了解得越多，尽早掌控它们的几率也就越大。

2.3.3 发布条件

要详细列举出项目产品的关键可交付物。想识别出它们，不妨问一问："要是不那么做，我们

还能发布产品吗？”要将功能、性能和质量要求都涵盖在内。要想了解更多细节，请查看2.4节。

2.3.4　目标

项目经理在撰写项目章程时，也许已经对目标有所了解了。在准备好制订项目计划时，也许对目标已经了解得更多了。如果在制定章程时不知道任何目标，这时应该好好想想了。

已知的目标也许隶属于以下几类。

- **产品目标**也许包括这样一些需求，它们已经被设定好优先级，但是不承诺在当前发布版本中完成。这个列表也许已存在于产品的待办事项中（请看16.6节）。
- **项目目标**也许是诸如性能标准之类的目标，对它们的要求会高于一般需求，或者是“在产品交付时，要将未解决缺陷的数目从50个减少到40个”。尤其是在管理一个工程的情形下，每个子项目的目标要特定于该项目所在的领域。项目团队要解决某些特定的技术债务，也许也可以作为项目的目标。
- **团队目标**可以是“增加产品的自动化冒烟测试所占的百分比”。团队也许希望改进某个特定功能的性能或可靠性。
- **组织目标**可以是“减少项目的耗费时间，以提升组织的敏捷性”。[①]

如果目标已经在项目章程中详细列出，在项目规划中就不必再写进去了。二者选其一，将项目目标明确出来即可。

2.3.5　项目组织

要明确说明团队在项目中的职责分配，指明项目经理如何使用生命周期组织项目工作，要采纳哪些关键实践，以及是否有决策人可以影响当前项目。如果你知道（或者怀疑）需要一些原型化的工作，要在这里说明。“史蒂夫和詹妮将会为那个功能集合搭建架构原型。比尔会进行同步的测试工作。因为时间紧迫，项目团队会选择可以让我们持续复查代码的技术。”

要说明项目的一般运作方式。比如，在项目启动时加强整个项目团队意识，招聘新人，开发包括代码和文档在内的完整功能，编写所有的代码，同时检查一下（在那个时间）可以记录些什么，诸如此类的事情。

此处的实际发生细节，取决于项目所采取的生命周期模型和团队的结构。要是有一个产品负责人专门负责作出与产品相关的所有决策，可以考虑是否要给他一个任命。如果有多个人同时决定功能特性以及彼此之间的权衡，就应该把这些决策人和他们负责的领域都列出来。

---

① 感谢Bill Ramos提供该示例。

如果使用的是受时间盒限制的迭代，要说清楚每个时间盒的长度是多久。如果使用版本火车的方式，要说明每列火车持续多久。

2.3.6　日程总览

应该创建一个日程总览，其中标有主要的里程碑，还要说明人们从这些里程碑处可以得到什么。如果使用迭代或增量式开发，要解释迭代（或增量）的持续时间，并说明在每个迭代（或增量）结束后可以预期得到哪些产出。不要把工作分解结构（WBS）放在这里，要是有的话，可以给出指示，让希望了解更多细节的人去自行查看。无论使用何种生命周期模型，如有beta测试或是早期的客户发布版本，要把这些作为里程碑对外说明。

工作分解结构（Work Breakdown Structure，WBS）：它是任务的组织形式，展示它们之间的依赖、持续时间和所有者等信息。WBS级别越高（在项目中出现得越早），能得到的信息就越少。WBS是会随着项目进度而演变的。

日程总览让我的领导知道项目的实际进展

——特里，项目经理

目前我正在管理第二个项目。领导们不相信这个项目会比上一个需要更多的客户参与。我编写了带有里程碑的项目总览，向他们说明情况。下面就是它大概的模样。

日期	里程碑
7月1日	项目启动。
7月15日	向露辛达展示Web界面的原型。（待续）
7月30日	向露辛达和厨房职员们展示厨房界面的原型。
8月15日	内部交付Web和厨房界面。
8月30日	向露辛达挑选的5个客户交付早期版本（beta测试）。
9月1日	开始beta测试。
9月30日	结束beta测试。
10月30日	系统上线，包括所有的客户订单和厨房的界面。

当我的领导们看到我在做什么之后，他们就意识到为什么我要提出对时间、人力和客户接触的请求了。

对于复杂的项目来说，如果在开始之间就已经感觉到日程安排上的风险，不妨考虑多准备几个方案，让出资人进行选择。图2-1中提到的组织就是基于矩阵的方式来组建项目团队的。

项目经理要确保自己可以理解项目的价值，而不是只看到不同方案的成本。如果交付的产品不能达到发布条件，其相关的方案也就不能提供足够的价值了。

不同方案	实际时间	架构时间	开发时间	测试时间	文档时间
功能1和功能2	2个月	1人月	6人月	6人月	2人月
功能1、2、3	4个月	2人月	10人月	10人月	3人月
功能1、2、3、4、5、6	12个月	6人月	60人月	50人月	8人月

图2-1　以矩阵方式给出项目团队的不同运作方案

2.3.7　人员配备

很多项目经理不能控制项目团队的人员配备。如果在项目开始的第一天就把所有的人都召集到位了，那么出现人员变动可别吃惊。如果需要从其他组或是团队中调动人手，要在这里说清楚：要在何时需要多少、何种类型的人员。请看第7章，以了解具体措施。

2.3.8　建议日程

项目经理要根据理解程度，列出主要的里程碑。如果有反映最初日程安排的甘特图，要在此处加入访问指示。当然，在重新规划后，要记得更新正在使用中的甘特图，以及时反映项目当前状况。将使用中的甘特图与项目的规划分开，这样更容易更新。

我喜欢使用黄色的即时贴来安排日程，所以在我的项目中很少使用完整的甘特图。我也因为说了下面的话而被大家所知："要想了解当前最新的WBS，去项目房间的墙上看看吧。"（如果没有项目专用房间供张贴甘特图使用，就要使用公共区域，来让大家都能看到日程安排。这样一来，如果日程有问题，项目团队也可以早点提醒项目经理。

波浪式规划方式（请看5.6节）用得越多，在第一波之后需要加入的细节就越少。

提示：小心过早细化日程

要反复修订项目日程，随项目进程补充细节。请看第4章，以获得更多信息。如果过早地向出资人提供了非常详细的日程，他们就会认为项目中几乎没有风险。他们会注意到日程中的项目结束日期，并以此作为项目真正的结束日期。

2.3.9　制订项目风险列表

在项目规划中，至少要将排名前十的风险记录在案。还要经常监控这些风险，并在适当的时机更新这个列表。如果觉得项目目前的风险不到十个，不妨跟项目团队一起坐下来，进行一次头脑风暴［DL03］。

要尽早开始识别和管理风险。图2-2是我从一个真实项目的启动过程中拿过来的风险列表示例。

风险序号	风险描述	发生概率	严重程度	暴露程度	反应时间	应对计划
为每个风险排定序号	用一个短语或一句话为风险命名	风险发生概率	如果风险发生，会造成影响的严重程度	发生概率乘以严重程度	应对风险的时间	处理风险的计划
1	除非到了项目后期，否则我们不知道算法的速度是否够快	50-50（中等）	高	（M,H）	7月14日	增加测试人员，与算法开发人员一起进行测试
2	启动超过2分钟	低	高	（L,H）	8月21日	在每次构建时监控启动过程

图2-2　风险列表

随着项目的进行，要更新风险列表。不妨使用第8章和第9章中的实践，来帮助规避风险。

2.4　制订发布条件

发布条件会告诉你项目中“完成”的含义（见［Rot02a］和［BWe01］）。有些项目经理会在下面这些状况下发布软件：某个资深经理说该发布了、测试人员“核准”了，或者是人们觉得时机成熟了。问题是大家对于“完成”的定义各不相同。

制订发布条件不是为了责怪哪个组的人，而是要为产品是否可以发布提供一些客观的评判标准。出资人和客户可以用发布条件作出合理的决策，对产品的质量和风险做出判断。

制订发布条件时，要先判断当前项目的关键因素是什么。例如，为什么公司要做这个项目？

为什么客户会买这个产品？

利用发布条件，项目经理可以将整个产品的职责分工贯穿到产品的发布之中。比如，销售人员能够卖出满足这些条件的产品吗？技术支持人员能够为这个产品提供支持吗？培训人员能否制订培训计划并展开相关培训？当项目经理与组织各个职能部门的人共同商讨“成功”的含义时，他们会意识到自己不只是完成了分内的工作，而且也为项目经理指明了成功的方向。

项目经理一旦知道了成功的含义，就可以定义这个项目最重要的因素——发布条件了。

使用下列步骤来制订和使用发布条件。

(1) 确定当前发布最重要的因素，这样可以将监控发布条件的活动贯穿项目始终。
(2) 草拟发布条件。
(3) 让发布条件符合SMART原则：确定的（Specific）、可测量的（Measurable）、可达成的（Attainable）、相关的（Relevant）和可跟踪的（Trackable）。请看2.4.3节。
(4) 获得项目团队与高层管理人员认可。

2.4.1　确定当前项目最重要的因素

当前项目的关键因素是由组织的需要和客户的需要共同组成的。客户在购买产品时，不会考虑其中有无缺陷或是缺陷数目，而是要看到这个产品解决了困扰客户的某个问题。在制订发布条件时，是应该考虑缺陷或缺陷水平，但是要确保发布条件中不仅仅包括缺陷相关的内容。想想你要为客户解决什么问题——无论是内部客户还是外部客户。

有时，发布日期是最重要的；有时，某个功能或一系列功能决定了项目的成败。通常来说，日程安排、功能特性和低缺陷率共同构成了项目的关键因素。就像1.2节中提到的，这些都取决于客户和他们的期望。

丽塔在一个依赖于现金流转的创业公司工作。公司的资金尚未完全到位，所以非常希望能早日交付产品，这样才能获得足够的现金维持公司生存。在过去的前三次发布中，唯一的发布条件就是发布日期，产品必须要交付给客户，公司才能在法律上确认产生收入。一旦公司成功熬过前几年，并且资金也全部到位后，就会加入其他的发布条件，诸如缺陷数目、测试过程、代码冻结状态（开发人员停止修改发布版本代码的次数）。

丽塔是这样说的：“当我们还处于创业阶段时，必须要保证把头伸出水面，维持生存。最开始的客户非常想要我们的产品，而且愿意一起工作。去年的版本发布后，其效果让我们都觉得非常惊讶。突然之间，客户开始关心缺陷问题了，他们之前从未如此关心过。管理层想知道我负责运转的测试组是什么样的状况，这使我压力倍增。唯一能让我舒心的，就是我已经与整个项目团

队和高级管理层事先确认过：早先设定的发布条件可以满足要求。但是我们没有意识到，当初没有完全弄清楚这个产品的需要。现在我们明白了，不能仅仅使用日期或是缺陷、测试数目来判断版本是否能发布。我们需要从大局出发，这样才能知道应该何时发布软件。

2.4.2 草拟发布条件

事先草拟一些发布条件是很有用的，这样就可以以它们为基础展开讨论。（如果有测试经理参与项目，项目经理可以请他完成或是与他一起草拟发布条件。）制订条件时，如果条件许可，要在上市时间和客户需求、缺陷状况、性能和可靠水平之间达成平衡。没有必要第一次就把所有的条件都订正确，只要能给大家一个讨论的基础就行。

在草拟发布条件时，要在纸面上注明“草案”字样。应该使团队成员和出资人知道这些只是供讨论用的条件草稿，而不是已经做出的承诺。

丽塔在开始时写出了一系列发布条件的草稿，供项目经理和团队成员讨论之用。下面就是丽塔的条件草稿：

- 所有代码必须针对所有运行平台编译并构建。
- 没有高优先级的bug。
- 所有未解决的bug和临时解决方案都要记录在版本发布说明中。
- 所有计划好的测试都要运行，要保证通过率至少为98%。
- 在最后三周内，未解决缺陷的数目要不断下降。
- 在发布之前，功能X要由开发人员完成单元测试，由测试组完成系统测试，由客户A和客户B完成验证。
- 准备6月1日发布。
- 所有未解决的缺陷都要由跨职能团队进行评估。

不过，当她与项目经理（下面简称PM）讨论后，丽塔意识到，她的初始条件并不完全符合客户或公司的要求。没错，客户很关注缺陷问题，但他们还没达到丽塔所关心的这种程度。实际上，只要几个主要客户对发布版本满意，其余的客户也就没什么大问题了。

刚开始的时候，对于丽塔能够从客户的角度出发看待整个发布，并能为整个发布提出一个照顾到各方平衡的方案，她的PM非常惊讶。他原本认为丽塔会更多地从传统测试经理的角度出发，力图提升质量，降低缺陷率。但是，丽塔从之前在公司做项目积累的经验知道：在对某次发布做决策时，低缺陷率只是考虑因素之一。

丽塔与PM和其他团队成员一起，修订了发布条件，并得到如下列表：

- 所有代码必须针对所有运行平台编译并构建。
- 没有高优先级的bug。
- 所有未解决的bug和临时解决方案都要记录在版本发布说明中。
- 所有计划好的测试都要运行，要保证通过率至少为90%。
- 在最后三周内，未解决缺陷的数目要不断下降。
- 在发布之前，功能X要由开发人员完成单元测试，由测试组完成系统测试，由客户A和客户B完成验证。
- 准备6月1日发布。

这些条件没有覆盖丽塔对于发布版本的全部要求，但是它们已经说明了对于当前发布的所有关键因素：发布日期、足够好的软件、一个已经完成测试并由两个客户验证通过的特定功能。测试组计划好的测试只要求通过90%，丽塔对此有些失望。不过在听到其他人的说明之后，她接受了这些发布条件，因为发布日期更重要。对于发布条件中不再要求在临近发布时由跨职能团队评估未解决的缺陷，丽塔也有些担心。不过，产品经理向丽塔保证，他已经跟PM讨论过这个问题，而PM将会把相关意见转达给市场和支持部门。

2.4.3　让发布条件符合 SMART 原则

在草拟发布条件时，要确保团队每个人对其的理解完全相同。我使用SMART原则——确定的、可测量的、可达成的（、相关的和可跟踪的——来检查条件是否合理、客观。T表示可跟踪性，这可以让整个团队明白：在创建发布条件时，我们要能够在项目的整个生命周期中评估这些条件。

对于当前项目的产品来说，每个条件都应该是确定的。如果一个条件是可测量的，就可以根据这个条件来评估软件。发布条件并不是那种必须要花些力气才能达成的延展性目标（stretch goal），所以要让每个条件都是可以达成的。通过评估产品是否符合客户和管理层的要求，来确保发布条件的相关性。如果条件是可跟踪的，那就可以在项目进行过程中评估条件的状态，而不只是在项目的最后一周才这样做。

“搜索必须在5s内返回第一组结果”是一个客观而且可测量的条件，可以对它展开测试。“所有未解决的缺陷必须由跨职能团队复查”是另外一个客观而且可测量的条件的例子。跨职能团队是否完成对未解决的缺陷的复查，其结果是确定无误的。

“性能要好”就是一个模棱两可的条件。要把它改成客观而且可测量的条件，应该这样描述：“性能场景A（对应于用例A）要在10s内完成。”要给出特定的场景，这样人们就可以引用它，并为其提供度量方案，团队也可以知道他们是否达成了性能要求。

2.4.4 在发布条件上达成多方共识

得到了合理的发布条件之后，就该取得各方面对它的共识了。如果有人对草拟的条件有负面意见（“不，我们做不到”），要找到他们担心的原因。产生发布条件的过程，也是揭示众人对于产品和项目的假定和忧虑的过程。在就发布条件获得大家共识的过程中，可以提下面这些问题：

- 在发布日期之前，我们是不是必须满足这个条件？
- 要是我们不能在发布日期之前满足这个条件，会发生什么？
- 如果不能满足这个条件，我们的产品或公司是不是会因此而承担风险？人们的安全感是不是会因此被破坏？

通过回答这些问题，整个团队就会更加注意当前发布版本的要求。

可以利用下面这些方式来获得大家的共识：草拟发布条件，针对其展开讨论，并在团队会议上就其达成共识；与整个团队草拟发布条件；与团队中的各个职能经理一起草拟发布条件。有了发布条件草案后，在项目团队会议上将其作为一项讨论议题。我喜欢与整个团队一起讨论并产生发布条件，因为这样一来，整个团队都会对其形成归属感，而不仅仅是我或管理层。不过，要是项目规模很大，或是团队之前从未用过发布条件，而且希望每个成员都能理解工作目标的话，还是先试着在团队会议之前草拟一份发布条件吧。

一旦获得多方共识，就可以用这些条件来监控项目了。

2.5 使用发布条件

到产品发布的时候，发布条件只能有“满足”或“未满足”两种状态。不可能部分满足某项条件——其实就是没有满足。在项目经理与高层管理者讨论软件产品目前的状况时，这种二元的方式是很有用的。如果只是说部分完成了，他们会认为你已经完全搞定了；如果说还没有满足某个条件，他们就会认为你还在努力工作。

在团队会议中，要跟团队一起讨论距离满足发布条件还要多久。整个项目过程中，在进行讨论的同时去看看项目仪表板（见第11章）的状态，整个团队可以评估当前项目的完成度是多少。如果项目有正式的系统测试阶段，可以将发布条件作为测试状态报告的一部分。

发布条件可以作为早期的警告信号，让大家知道团队可能无法准时发布当前版本。曼尼是一个项目经理，他所带领的六个月的项目目前进展到一半。曼尼评估了距离发布日期的进度，担心团队无法按时交付。他打算跟团队一起制订发布条件，让大家更加了解这次发布要达成的目标。

在接下来每周的项目团队会议上，曼尼用发布条件来确认项目在不断取得进展。这样连续做了几周，效果都不错。直到有一周，在评估发布条件时，一个工程师说："我做不到。我试过很多种方法，看来还是无法在交付日期之前满足这个条件。"曼尼说："那好，我会去跟上面说说看能做些什么。在去之前，还有人觉得哪些条件不能满足吗？"另一个工程师说："在发布之前，我无法让性能做到如此优秀。咱们当初讨论发布条件的时候，我觉得是没问题的，但现在我知道那是无法做到的。"

使用发布条件，曼尼可以得到项目进度的一些早期数据。对于这个团队来说，在进行到三分之二的时候知道他们无法达成发布条件，总好过在最后一刻才发现问题。项目经理在了解到项目的真实状况后，可以就此与管理层沟通，看看在哪些方面进行折中，可以产生最现实的结果。在这个例子中，为了让产品满足发布条件，曼尼可以就发布日期进行重新谈判。

理想状况下，项目经理可以随着项目进度评估发布条件，并在项目预期结束时满足这些条件。不过人生不如意十之八九，项目也是如此，发布条件不可能总是能够达成的。发生类似情况时，要坦诚面对现实。

在必要时变更发布条件

使用发布条件，可以让项目关于"完成"的定义保持稳定。不过，在下列情况发生时，项目经理要考虑改变发布条件：进一步了解了这个项目中"完成"的含义时；认识到无法在预定发布日期前满足所有发布条件时。

如果进一步了解了"完成"的含义，不妨问问自己之前关于发布条件的问题：我们是不是必须要满足发布条件？要是没有满足发布条件，会发生什么？请看2.4节。

假设项目经理所在的项目无法满足发布条件。首先，要跟团队确认为什么无法满足条件。接下来，向管理层解释无法满足条件的原因。促使管理层向项目团队解释目前的状况，就像这样："我们曾经认为这些条件很重要，但是现在知道发布日期是更重要的。我们将会准时发布产品，即使不能满足所有的发布条件。"如果事实如此，项目经理可以让他们补充："我们会找出这次不能达成发布条件的原因，并在下个项目时注意不再发生同样的问题。"如果不向团队解释，他们会觉得自己正在玩日程排定游戏，请看6.1节。

铭记在心

- ❑ 项目规划是在不断进行的，这只是开始。
- ❑ 为项目团队、出资人和项目经理自己制订发布条件，以明确定义"完成"的含义。
- ❑ 项目规划不必完美无瑕，但是它必须存在。

第 3 章

使用生命周期组织项目

假设你打算参加一次朋友聚会，路上要花一天的车程。你已经事先规划好了行程，并在地图上标注了目的地。可是刚钻进汽车，一个朋友就打来电话，说有一段路被水冲垮了，现在正在抢修，你得绕过去。最终，你还是到达了目的地，不仅安然无恙，并且还及时赶上了晚餐，但是走的路线却与当初计划的不同。

选择项目生命周期的思考过程也是如此。一旦踏上项目之路，没有哪种生命周期是完美无缺的。可能要对其做些拓展，并且不时修修补补。生命周期是组织项目的理想化方式。有些生命周期可能更适合你的团队（和项目）。如同会出现道路被水冲垮或是遇到障碍，所选的生命周期模型要保证足够灵活。

3.1 理解项目生命周期

生命周期是项目经理和团队组织产品开发的方式。定义需求、设计、开发、测试以及与这些工作同时进行的过程，都算是生命周期的一部分。你可以选择阶段–关卡式开发，或是采纳迭代的方式。可以规划一个正式的设计阶段，或是让架构和高层设计随时间演化。可以在项目进展中加入测试，或是在临近结束时完成所有的测试。可以先进行原型化的工作再填充功能，或是逐个实现功能，同时观察架构的演化。

从整体上组织项目时，不要把现实状况理想化。即使曾经遇到项目需求不完整的问题，也不要在规划时就希望先产生完整的需求。可以选择一种生命周期，让它能随着项目推进不断发现新需求。要讲求实效——将风险记在心间，选择的生命周期要有助于识别项目风险，帮助项目经理交付成功的产品。

很少有项目能够沿着从需求到发布这样一条直线推进，也没有多少项目能够规规矩矩按计划进行。即使项目经理最熟悉的阶段–关卡流程[①]和瀑布式生命周期，它们也不是组织项目的唯一方

① 阶段–关卡流程（stage-gate）模型是由罗伯特 • G • 库珀（Robert G. Cooper）提出的，是一种高效管理产品开发流程的方法。它是新产品开发的蓝图，涵盖了从开发理念到产品生产的整个过程。它有一系列预设的步骤，即阶段（stage）。每一个阶段都包含一整套跨功能的、平行的具体任务，在被获准执行下一个阶段以前，必须顺利完成现阶段的任务和活动。每一个阶段的入口被称为关卡（gate）。这些关卡通常是一些正式的会议，对流程进行控制，并发挥质量控制、检查站和下一阶段行动标志的作用。——译者注

式。项目经理在进行规划时，要看看选择的生命周期能否处理项目的风险。如果风险与生命周期不匹配，不要强求它们能吻合，还是选择另外一种生命周期吧。

在思考某种生命周期是否实用时，不能仅仅考虑内部的项目风险。客户和他们的期望也是风险的一部分。在思考如何组织项目时，要记得考虑客户的要求。他们需要什么？他们有哪些经验？他们愿意投入多少精力与项目团队一起工作？（请看1.7节。）

如果在规划时只需要考虑一种客户，而且也只有一种驱动因素能够影响产品，那该多好……但是现实很少如此。项目经理必须找出对于客户最重要的风险。选定的生命周期要可以很好地应对下面这些风险：是否能够及时发布、缺陷、特性、成本等，其他的风险可以通过一些实践来管理，还要为项目选好人手。

3.2　生命周期概览

不同的生命周期有不同的风险处理方式。现在主要有四种不同类型的生命周期：顺序式、迭代式、增量式、迭代/增量式——此后我会称之为“敏捷”。（实际上，要使用迭代/增量式生命周期，你不必完全遵循敏捷宣言[①]中的敏捷价值观。不过，要想让敏捷生命周期发挥作用，不遵循敏捷价值观几乎是不可能的。）要想了解各种生命周期如何管理项目中可能出现的风险，请看图3-1。

如果还是无法区分这些生命周期，请看图3-2。用甘特图表示不同生命周期，各个类型的生命周期如图3-2中所示。

在顺序式生命周期中，团队应该首先获取所有的需求。基于这些需求，团队可以进入分析和设计阶段，以决定系统的整体布局。大家就整体布局达成一致意见后，团队进入开发阶段。开发完成后，团队将会整合所有的功能，再开始最终的测试。在现实状况中，当前阶段尚未完成时，下一个阶段就可以开始了。不过，“一次只做一个阶段该做的事情”的心态在顺序式生命周期中是很常见的。

顺序式生命周期会花费较长的时间。人们觉得用这种生命周期，可以预测工作持续时间，比如实现功能需要多久、找到并修复缺陷需要多久、整合系统功能需要多久、管理需求变更要多久，等等，但这些事情本来就是难以预计的。除非项目经理有一个可以预测工作的水晶球，否则不可能估计到未来要知道的每件事情。在顺序式生命周期中，项目经理要允许占用计划外的时间，比如项目结束时的系统测试，以弥补项目过程中的未知风险和问题可能造成的损失。

---

① 请看http://www.agilemanifesto.org。

生命周期类型	该类型生命周期范例	优势以及成功的必要条件	项目优先级	成功预期
顺序式	瀑布、阶段-关卡式	管理成本风险（如果管理层采用阶段式） 需求已知并已就其达成共识 系统架构已被深入理解 项目需求不会发生变化 项目团队不会发生变化	1. 功能集合 2. 低缺陷率 3. 发布时间	成功，并可得到反馈
迭代式	螺旋，不断演化的原型	管理技术风险 不断演化的需求	1. 功能集合 2. 低缺陷率 3. 发布时间	迭代中的任务已做规划，并且按计划完成
增量式	项目日程由设计决定，并按阶段交付	管理日程风险 可以应对小的需求变更，但是对于会影响到架构的变更无法处理	1. 发布时间 2. 低缺陷率 3. 功能集合	成功
迭代/增量式	敏捷（例如Scrum、XP）	管理日程和技术风险 如果团队成员不在同一物理位置，人员职能不完备，就难以实施	1. 发布时间 2. 功能集合 3. 低缺陷率	成功
其他	编码并修复		1. 发布时间 2. 功能集合 3. 低缺陷率	不可能成功

图3-1　不同类型生命周期管理风险的方式

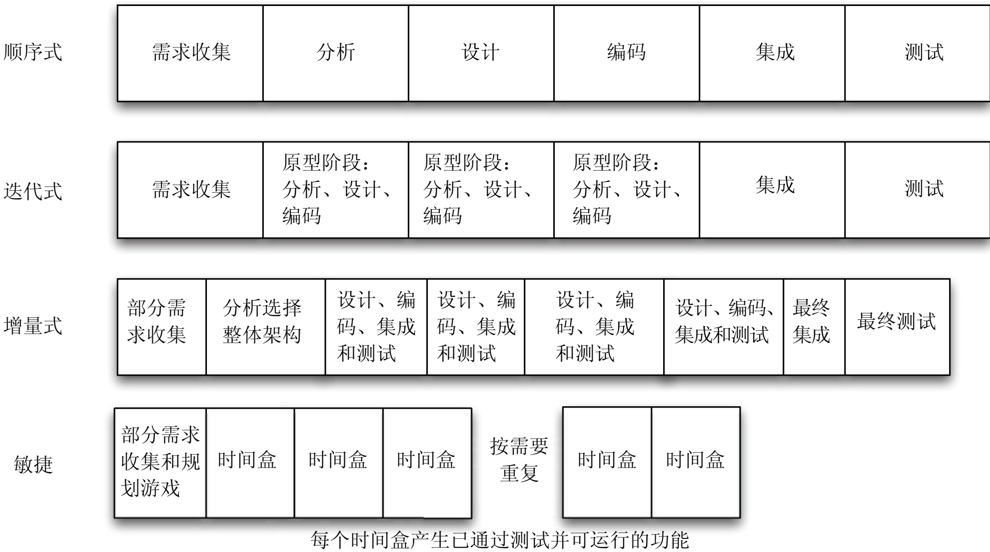


图3-2　用甘特图的方式看不同的生命周期

使用迭代式生命周期要创建系统的部分原型，帮助团队了解原型，并试图以此解决一些预测问题。在迭代式生命周期中，项目团队会在每次迭代中开发产品的一个部分。至于是否必须在迭代结束时完成产品某部分的开发，绝大多数迭代式生命周期模型对此都不会作要求。迭代式生命周期不要求同时进行测试和集成。（当你在构建架构原型时，无法对未来有很好的预测，除非已经开始构建和集成功能。）

在需求收集和分析阶段，增量式生命周期与顺序式生命周期有类似之处。不过，成功的团队会用时间来限制需求的收集和分析。不久他们会按照功能分成不同的团队。每个团队每次开发一个功能，完成测试和集成后，再开始开发另一个功能。使用增量式生命周期的项目，在进行增量式开发之前，也会遇到预测不准的问题。进行增量式开发后，开发团队会得到使用增量式生命周期构建功能的反馈，从而决定项目的走向。

敏捷生命周期混合了迭代式和增量式这两种生命周期，但是迭代更短，增量更少。用敏捷生命周期组织启动一个项目，只需要一点点前期的规划工作——只要足以启动项目，而且知道产品负责人对当前发布版本的期望就可以了。产品负责人甚至可以将他们需要的功能划入到指定的迭代中，但是团队不会花很长时间做版本规划。实际上，他们会去规划一个有时间限制的迭代（一至四周）中要完成的工作。团队在这些时间盒中开始工作，首先实现最有价值的功能。他们会收集很多数据，比如他们的工作进展速度、问题解决情况、需求理解状况，等等。随着项目的推进，团队从产品负责人那里获得关于产品功能的反馈。他们会基于团队的工作效率和环境变化，为后续迭代重新进行规划。由于团队会主动寻求对于工作和流程的反馈，这种生命周期能够将收集到的反馈与项目进展完美地整合到一起。这些反馈包括项目的真实状态、开发效率、寻找和修复缺陷的效率以及团队的假定，等等。

除非项目团队主动按照功能去规划开发、测试和集成的工作（正如在并行工程或敏捷中所做的），否则他们一般会遵循“设计—编码—测试—调试”这样的循环，如图3-3所示。开发人员首先设计产品，接下来进行编码，然后测试，再进行调试。这个过程可能花费数周甚至数月。试着叫一帮开发人员一起开发某个产品，你就会知道需要多久才能得到完成测试的产品代码了。

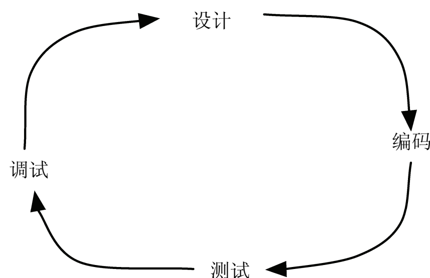


图3-3　“设计—编码—测试—调试”循环

提示：编码并修复——永远糟糕透顶的生命周期

绝不要选择“编码并修复”作为生命周期。绝不！绝不！！绝不！！！有些人直接使用这种的方式，是因为他们觉得这样做更快，不必花时间做规划、开发原型或是收集需求。不是这样的。无论选择哪种生命周期，在项目开始时都要做规划，哪怕是一点点也成。呆伯特对这种生命周期有极佳的概括。尖头老板说：“你们就开始编码吧，我去搞定需求。”

在项目开始时，是不必获得所有需求的。按照功能进行实现，用时间盒限制迭代，与客户一起工作，这些措施可以确保开发的产品符合客户的要求。但千万不要觉得可以在没有客户的情况下进行开发，或是不进行规划就启动项目。你可以这么做，但是项目要想成功就势比登天了。

3

生命周期是组织项目的理想化方式。选择了某种生命周期，并不意味着就要僵化地执行下去。项目经理可以根据项目的风险情况，整合其他生命周期的管理方式。

举例来说，如果项目经理遭遇过必须提早确定架构所带来的风险，而且知道目前收集到的需求只是一部分（还不能使用敏捷生命周期），那么图3-4可能会对此有所帮助。使用这种结合方式，项目团队可以先在迭代中做一些原型化的工作。了解到足够情况后，团队可以按照进度进行增量式的开发、实现、测试和集成的阶段。

图3-4所示是一种组合式的生命周期，用到了迭代式生命周期的一部分（原型化）、增量式生命周期的一部分（3个完整的功能实现和后续的开发），其中有些想法来自敏捷社区（对需求的初步了解，并随着项目进展将需求迭代化），项目结束时的最终测试用来管理风险。

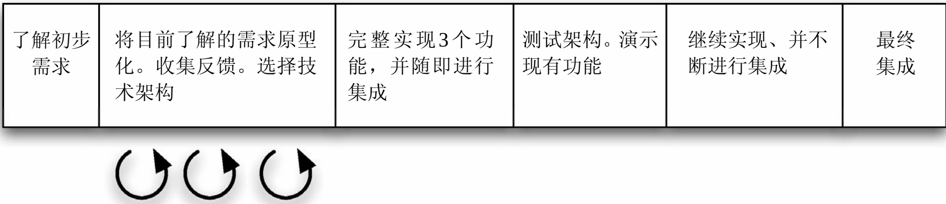


图3-4 迭代式和增量式生命周期的结合

3.3 在项目中寻求反馈

开发人员首次获得代码的反馈，是通过测试完成的。而且，如果测试没有集成到项目之中，这些反馈要很晚才能出现。

开发人员得不到反馈，项目经理就很难评估剩余的项目风险、项目的状态以及团队的开发

工作效率。很难知道工作是不是真的完成了。缺少反馈，也正是顺序式生命周期被称为“预测式”生命周期的原因。顺序式生命周期预测未来，但是缺少足够的数据，无法检查现有工作能否作为未来工作的基础。团队越早获得产品反馈（不是产品说明，虽然这也会有帮助），预测就越容易。

在敏捷式生命周期中，仅仅查看类似甘特图的图形，不可能看到敏捷生命周期提供的反馈循环。在每个时间盒中有两个反馈循环。常规的反馈循环如图3-5所示。开发人员边写代码边测试，这样可以马上得到反馈。每天和每个迭代的反馈过程如图3-6所示。在开发人员和项目经理之间的反馈，是敏捷生命周期能够成功的根本原因。

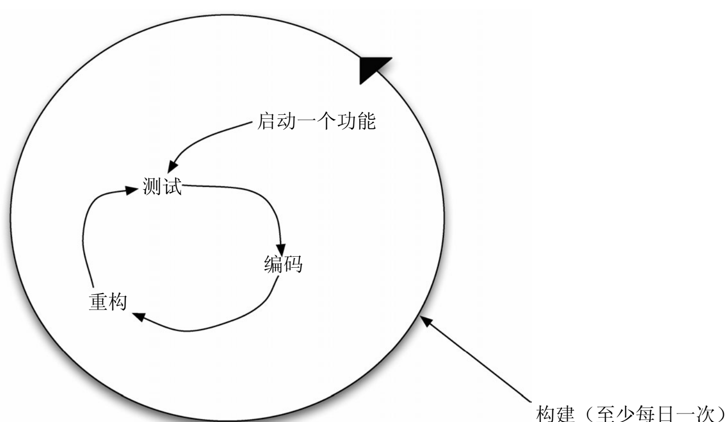


图3-5　“测试—编码—重构”循环

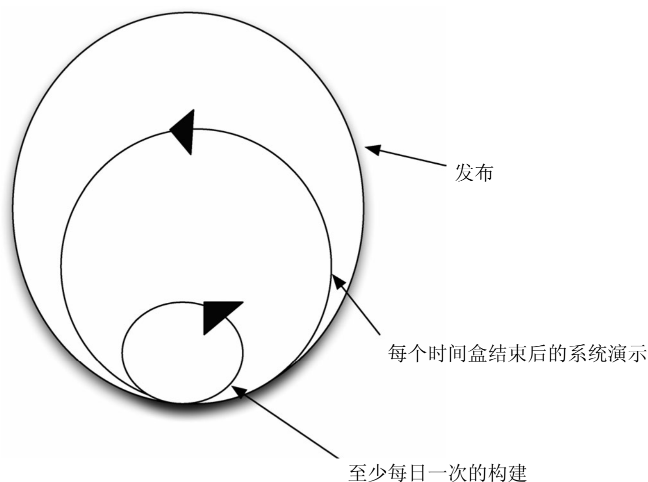


图3-6　敏捷生命周期如何收集反馈

3.4 大规模项目需要组合使用多种生命周期

没有哪种生命周期可以满足所有人的要求。对于使用大规模团队或是在多个地点展开的项目，可能每个团队都使用自己的生命周期。（要了解更多信息，请看第12章。）比如一个有60名开发人员和20名测试人员的项目，其生命周期如图3-7所示。

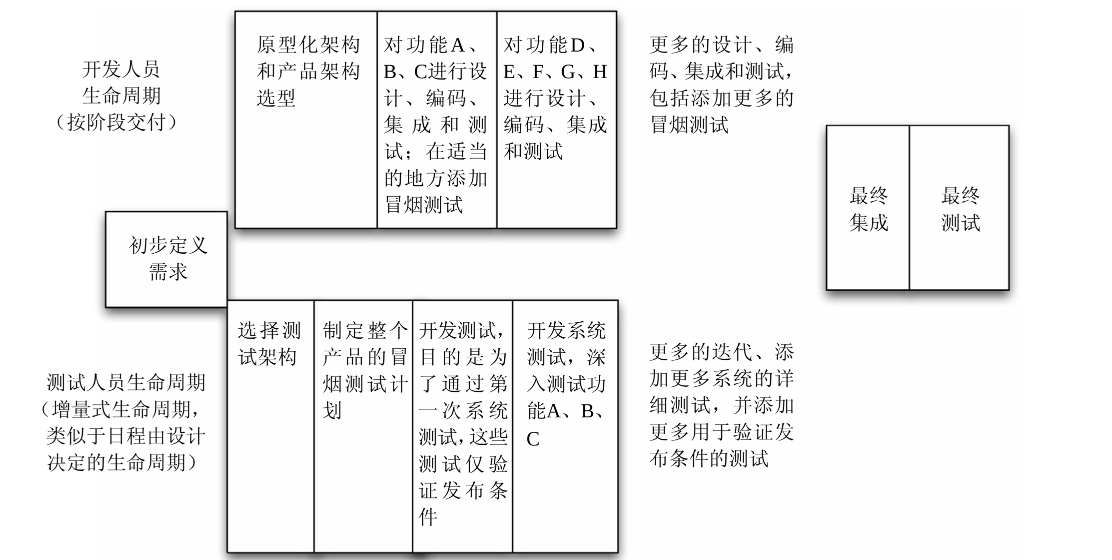


图3-7　一个大项目的生命周期：多种生命周期的组合①

开发人员使用增量式的生命周期。测试人员使用不同的增量式生命周期，以此种方式展开测试，他们就可以通过广度优先的方式来评估系统状态。开发人员会一个接一个地完成各个功能特性的开发并集成，这样一来，即使管理团队要求早日交付，也不会出现问题。

我曾见过在多个地点展开的项目，他们组合使用了多种生命周期。一个地点的开发人员会使用演进式的原型化方法，另一个地点的开发人员则按阶段交付。这个项目长达18个月。大家同意在项目开始以及每隔三个月的时候，讨论下一阶段的交付物会是什么样子，并制订出这些交付物

① 读者可能注意到：此图中将测试人员使用的生命周期说明为增量式生命周期，可在图的右下方又注明“更多的迭代……”。就此，作者特意说明：此处所指的迭代并非之前所描述的迭代生命周期。首先测试人员会尽量测试增量的功能（也被称为chunk），当他们有时间时（类似于迭代）会去开发更多增量。整体的生命周期还是增量式的，但是测试人员只要有时间，就会向现有测试中加入更多测试用例，此时类似于迭代。［原文：It wasn’t a formally described iterative lifecycle. The testers did what they could to test a chunk (that’s the incremental part). When they had time (that’s the iterative part) they developed more increments (that’s the incremental part). The overall lifecycle is an incremental one. But, the testers added more as they had time to tests that already existed (the iterative part).］
——译者注

的发布条件。

另一个项目横跨三个国家。其中一个国家的开发人员使用两周一次的迭代，第二个国家的开发人员使用阶段式交付，而第三个国家的开发人员则使用四周一次的迭代进行开发。他们每个月定义一次可交付物以及交付验收条件，而不仅仅是在项目启动时进行。多站点项目管理的关键，是要定义出各阶段交付物的模样，以及何时完成交付。请查看12.4.2节。

包含硬件的项目

如果项目的目标是开发新一代的芯片或电路板（新产品研发），公司可能会提出一大堆雄心勃勃的需求，而硬件团队可能根本无法在给定的时间内完成。在明确知道硬件可以完成哪些功能、软件需要完成哪些功能前，项目经理要耐心等待，直到硬件制造完成。软件上的小问题，一般通过更新软件就可以完成修复。硬件就没有这个选择余地了，所以在时间和成本上的约束都会与软件项目不同。在硬件交付后再对其修复，其代价就是软件要适应硬件的日程安排。使用按阶段交付或敏捷生命周期组织这样的项目，软件团队可能会处于等待阶段，这时可以按功能逐个实现，或是使用存根（stub）来完成部分可以预先完成的工作。知道了硬件的功能范围后，有可能需要用硬件或驱动程序调用来替换这些存根，或是使用软件替换这些存根。在硬件部分达到一定的完成度之前，项目经理要明确了解包括硬件部分在内的整个项目的进展顺序，除非已经知道了底层功能的具体开发要求。你不需要等到所有的硬件都齐备，但你的项目中有“某些阶段的事情”是需要等到硬件完成才能做的。不过项目经理不一定非得等到硬件开发完成。成功的软硬件结合项目需要组合使用多种生命周期。

固件被视作软件还是硬件要看具体背景。如果构建的固件处于某系统之外，要被置入或下载到系统之中的，就可以将其视为软件；如果构建的固件要被嵌入到系统中而且不再改变，就应将其视为硬件。

如果集成的系统需要包括其他厂商的组件，项目经理就必须结合使用多种生命周期，并以稳定的方式来管理项目风险。举个例子，你要开发一个具备最尖端技术的电冰箱，它可以直接从网上订购日常食品，而且能够查看易变质食品的当前状况。

你可能要从Acme Analysis公司购买牛奶分析器。要从Meats-R-Us公司购买肉类分析器，从Veggies for Life Systems公司购买蔬菜分析器。低档的冰箱只有牛奶分析器，而高档冰箱中有肉类和蔬菜分析器。这几个不同的项目，要作为高、低档冰箱这个大工程的一部分（请看第14章）。

在约定交付日期，甚至是决定使用哪种生命周期之前，要与厂商商定项目中间的可交付物。使用不同厂商的组件进行系统集成，风险很大。要管理相关风险，不仅仅是选择生命周期这么简

单，还得随着项目推进为集成工作做相关规划。如果分析器厂商们必须每个月发布一次产品（即使这些发布不面向最终客户），那么就可以使用迭代式或敏捷的开发方式来管理风险。

如果项目经理发现对于所管理的系统集成项目，公司没有与厂商商定中间可交付物，项目经理可以考虑从现在开始与那些厂商讨论。不过即使厂商不同意，还是仍然可以使用增量式或迭代式方式来组织项目。

要想管理集成中最难应对的部分，我强烈建议进行增量式构建，要么使用增量式生命周期，参见图A.5，要么使用有时间盒限制的迭代（管理风险的最好方式），参见图A.6。不要使用顺序式生命周期，因为无法在这个过程中看到技术上的风险，只能在结束时才能发现问题真正所在。

3.5　管理架构风险

架构风险主要是指团队选择的架构能否满足当前项目的要求，没有哪种生命周期可以完全应对产品架构相关的风险。在架构代码开发完成并集成完毕之前，无法断言设计的架构能否正常发挥作用。我曾见过很多选择了顺序式（瀑布或阶段式）生命周期的项目团队，他们觉得顺序式的生命周期可以控制架构上的风险，但很不幸，他们错了。因为在瀑布生命周期中，最终的集成和测试是最后发生的。瀑布式和阶段–关卡流程对于架构风险的控制是最差的。

如果项目经理必须使用顺序式生命周期，又想早些发现架构风险，可以采取下面这些方法。

- 对于接近“完成”的原型，要尽早在项目中对其展开迭代，包括对这些原型进行测试。如果采用的都是“快而脏”的原型（就像在螺旋式生命周期中采纳的方式），那就很难发现架构的性能或可靠性是否可以满足实际要求。
- 尽早实现几个可以考验架构承受能力的功能。我喜欢用不超过三周的时间来实现这些功能。从实现这些功能的过程中，看看能否催生架构的新想法。同时要注意功能团队遇到了哪些风险，这有助于项目经理判断是否需要选择另一种生命周期。

  假定项目经理为了检验推荐架构的可行性，选择用三周的时间来实现四个功能。三个星期过后，团队发现这个架构只对两个功能的实现很有帮助。对于自己在三周的迭代中所能完成的工作量，他们的估算是不准确的。项目经理可以继续使用有时间盒限制的迭代来完成剩下的工作，或者选用不同的架构来重复这三周时间要完成的工作，看看不同的架构能否控制开发风险或提升开发效率。

  在顺序式生命周期的早期，要多做实验（即使这种生命周期不需要实验过程），这样在项目后期遇到架构或设计风险的可能性就越小。
- 用时间盒来限制整个架构相关的工作。让架构师和开发人员开发出至少三种不同的架构选择，他们还要告诉项目经理每种架构的长处和风险所在。

要想完全掌控架构风险，只有基于它实现某些功能并进行测试。任何Microsoft PowerPoint演示的架构［SH06b］（只在PowerPoint中实现的架构）都只不过是装点门面（window dressing[①]）而已，而且浪费时间。

项目经理所在的组织，或者存在出资人、资深管理层或PMO项目启动之前希望先看看架构（或者就是架构示意图）的情况，或者存在如果不进行架构复查可能会带来很大风险的情况。这种情况下，我仍然建议项目经理要完整地完成一些功能的原型化设计，这样多少可以获得一些经验，有助于判断能否放心地在项目中使用这个架构。

在项目中，项目经理要随时注意架构能否满足项目要求，而且这方面的风险发现得越早越好。如果不能及时察觉，整个项目可能因此而功败垂成。

3.6　从瀑布中摆脱出来

如果深陷顺序式生命周期的泥潭，可以用下面这些方法来尽量脱离困境，更灵活地适应现实。

- 用迭代来规划所有的工作，包括规划、需求收集和原型化等工作。
- 将产品原型化，并尽早向客户或其代理人展示。越多地从他们那里得到反馈，项目经理的日子就越好过。
- 从项目一开始就加入测试的工作。与测试人员一起，在整个系统可用之前就为开发人员提供反馈。
- 功能要逐个实现，完成后随即进行集成和测试。
- 如果必须提供文档（在顺序式生命周期中，文档是每个阶段结束的典型里程碑标志），要确保文档不是唯一的可交付物。与客户一起探讨产品原型，并交付可工作的产品，这样可以为项目团队提供很有价值的反馈。

传统的顺序式生命周期不会用到上述手法，但是它们对于项目的成功很有效。项目经理如何管理项目，要向出资人坦诚相告，告诉他们你要应对很多风险。处理项目内部的事务，可以像做香肠一样。项目经理不必告诉外人如何处理项目内部各种事务，就像做香肠的人也不会告诉你他们是怎么灌装、晾晒香肠的。

3.7　我最钟爱的生命周期

尽早向代码库中提交功能代码，我对此情有独钟。以我的经验看，除非团队完成了几个功能

① 感谢杰瑞 • 奥宾（Jerry Aubin）提供这样的说法。

的开发、测试、集成，否则架构是无法确定下来的。我喜欢在项目中使用Scrum来产生演进的架构和交付功能。这个项目管理框架可以让项目一目了然、易于审核，并且适应性很强。如果可能，我还会加入极限编程（XP）（请参阅［BF01］和［JAH02］）实践，或者判断架构的可行性，再使用诸如阶段式交付这样的增量式生命周期，进行功能交付。

如果团队能够以管理敏捷项目的方式进行协作[①]，而且也愿意这么做，敏捷生命周期模型就是我的首选了。不过，要是团队无法得到客户足够的投入，或者团队成员对于敏捷式的协作毫无兴趣，我还是倾向于使用阶段式交付的生命周期，并用时间盒来限制需求收集和架构设计阶段。（客户参与度不够的项目总是会出问题。相对其他类型生命周期，敏捷生命周期可以让问题暴露得更早。）

就算是很小、很简单的项目，我也很少使用瀑布式的方式。即便采取了螺旋式生命周期，我也会让团队进行类似阶段式交付的方式，这样我就可以得到已经完成了的工作，而不是一件件半成品。

铭记在心

- 在组织项目时，使用任何生命周期或是多种生命周期的组合，都可以让项目踏上成功之路。
- 不要怯于创建反映你自己项目实际情况的生命周期。“完美的”生命周期只是模型而已，你是生活在现实世界中的。
- 阶段-关卡流程或瀑布式生命周期，只有在确定使用它可以获得成功时才使用，而不是不经思考，上来就用。

① 请查看http://www.stsc.hill.af.mil/crosstalk/2007/04/0704Derby.html和http://alistair.cockburn.us/index.php/Agile_software_development:_the_people_factor。

第 4 章

安排项目日程

规划和安排项目日程是两种不相干的活动。在第2章中，你已经启动了项目的规划。接下来，应该考虑项目的日程安排和工作估算了。在组织项目日程安排和重新估算工作时，可能要修改规划，这很正常。最初的规划只要能让项目启动就可以了。在安排（以及重新安排）日程时，要准备改进规划。在重新规划时，如果需要改进日程，也不必担心。

4.1 注重实效的项目日程安排

对于启动项目来说，当初的规划已经足够了。接下来的日程安排，只要能把项目启动起来，也就行了。既然知道项目会随着时间演变，就没有必要详细安排整个项目的日程了。如果客户在签合同之前希望看到完整的项目时间表，要告诉他们：这个时间表只是基于目前状况所做出的最好猜测，是会变化的。还可以让他们先看看第5章中的提示：**估算要准确，而不是精确**。

几年前的一次会议上，我跟一位项目经理有一次谈话。我提到制订项目日程总是要花半天到两天的时间，而我想把两天缩减到半天。

这时，站在一旁的另一位项目经理开口了。她不相信我可以在半天之内完成项目的日程安排工作。我解释说我不会试着去安排所有的工作，而是只思考接下来大约一周的事情。然后我会构建主要的里程碑，并对后续几周的工作进行波浪式日程安排（见附录B）。

“你怎么知道结束时间是什么时候？”她问道。

“我不知道，至少了解得没那么精确。在没有任何数据的情况下，如果想过早知道结束日期，我一定会犯错的。既然知道一定会出错，那为什么还要花时间去做详细安排呢？”

她说：“天哪，我以前从来没这么想过。”

安排项目日程有很多种方式。我选择自顶向下的方式，所以我草拟了第一个计划，它可以帮我草拟出第一个项目日程。有的项目经理会先制订出一个项目日程安排草案。怎么样做让你觉得

最舒服、对项目最合适，就不妨那么做。但是项目计划或日程安排这二者都不可或缺。每个项目都需要这两个东西。

提示：对于项目来说，规划和日程安排二者不可或缺

一个注重实效的项目经理，为了先让项目启动起来，会做恰到好处的规划，并随项目进展不断进行规划和重新规划。不管怎么做，规划与日程安排都不可忽视，尤其是规划中的发布条件。在安排日程时，也许没有必要使用设计精美的甘特图，对于大部分项目来说，在墙上贴黄色的即时贴就足够了。但是规划和日程安排都必须要有。

项目的日程安排会与选择的生命周期有关系。不过要记住：生命周期是项目的模型，让别人看到项目是如何组织的。在创建日程时，可以将生命周期用作指导方针。不过每种生命周期模型都有其内在风险，无论采取何种方式安排日程，都要保证处理这些风险。对于阶段–关卡式的日程，可能要安排一些有时间限制的迭代和增量式的开发过程，还得做好不断重新规划的准备，对于某些项目来说，这样的开发方式是有意义的。要记住，生命周期可以作为指导方式，但不是严格的限制条件。

日程安排和工作估算是两种不同的活动。日程排定要安排工作任务的顺序，展示它们之间的依赖关系。工作估算则要猜测某个任务要花费的工时数。这两种活动有关联，如何安排日程，可能依赖于某项任务的估算工时数和需要的特定人员。

我也希望能轻轻挥舞魔杖，然后对大家说：“这里是对项目进行日程安排和估算的唯一正确之路。”但是我不能。我们总是要估算一些之前从未做过的事情。

对未知事物进行判断，这仍然是一门艺术。不过，要是知道自己采用了哪种生命周期，组织并安排日程就会变得容易了。

提示：用时间盒限制初始规划

前期规划活动耗费的时间要尽量少，特别是在团队人员已安排到位的情形下。只要产生的规划足以让项目启动就可以了。用一个小时制订项目章程，再用一个小时完成项目规划，日程安排的草案也用一个小时完成。使用时间盒的方式，可以让每个人把注意力放在项目启动的关键活动上。一旦大家知道将来的一到两周内要做什么，项目经理就可以再考虑规划和日程安排中还需要补充哪些内容了。

4.2　可供选择的项目日程安排技术

在安排项目时，我会从下面这几种日程安排技术中选择：自顶向下、自底向上、由内及外、

哈德逊湾式启动和短期迭代。

4.2.1　自顶向下式日程安排

自顶向下式的日程安排通常从设置里程碑开始。顺序式的生命周期经常采用自顶向下式日程安排，因为阶段都划分得很清楚。（提示，如果必须使用顺序式生命周期，要确定使用基于交付的规划技术来产生里程碑，见4.3.6节。）

将项目日程分为阶段、迭代或是几大块。把它们画在白板上，或者用即时贴粘在墙上。德怀恩·菲利普斯（Dwayne Phillips）推荐另一种低技术含量的日程安排技术——使用放在墙上的卡片。使用放在墙上的卡片安排日程，每个人就会把应该由自己完成的任务写在卡片上。然后再用线把这些卡片连起来［Phi04］。如果不知道从何下手，这个技术可以助你一臂之力。

团队会从最高级别的里程碑来组织日程，并制订任务支撑这些里程碑。如果有团队成员更深入理解了每个里程碑的含义，他们就会将里程碑拆分为子任务。

最底层的任务越小，估算任务完成需要的时间也就越容易。

4.2.2　自底向上式日程安排

自底向上式日程安排从特定任务开始。如果使用增量式生命周期，用自底向上的方式应该还不错。“我们知道应该先实现这个功能，然后实现这些功能，接下来我们就可以决定是否要继续开发了……”

不管是独自工作还是身处跨职能的团队，项目团队成员会从任务中产生里程碑。作为项目经理，你应该问问里程碑跟任务是如何组织在一起的。（你的技术水平越高，就越能帮忙。如果你没有任何产品的领域经验，那还是别打扰别人了。）

4.2.3　由内及外式日程安排

要想做到随机应变、应付自如，用由内及外的方式来安排项目日程就最合适不过了。在我举办的一个项目管理研讨会中，一位项目经理说：“在安排日程之前，我会先画一个思维导图，把我知道的所有与项目有关的东西都放进去。我也许了解一些可行性审查点，也许明白某些功能是怎么回事。但是项目中的很多事情纷繁复杂，很难立即判断出来到底需要多少时间。所以在开始安排日程之前，我希望把知道的所有东西都整理出来。”

思维导图对于创建人来说可能很清晰，但是项目涉及的其他人对其却不一定了解。如果这些

人在创建思维导图时就在场，对其的理解程度远比后来直接看到结果要好得多。如果项目经理及其团队使用由内及外式的日程安排，要保证团队在一起制订任务和里程碑。

4.2.4　哈德逊湾式启动

设想你所管理的项目对于你和团队来说完全是从未经历过的。你也不知道所在的环境是不是支持可用的工具，从何入手估算项目工作也毫无头绪。不妨考虑使用短期迭代来开始工作，比如“哈德逊湾式启动（Hudson Bay Start）”。

“哈德逊湾式启动”方式源自17世纪加拿大东北部的哈德逊湾公司，这家公司配备了运送皮毛的商船。为了确保商船不会忘记需要的东西，他们会在距离哈德逊湾几英里的地方先临时停留一段时间。由于离海湾并不远，商船可以确保他们在离开文明世界进入茫茫大海之前不会忘记任何工具和给养。使用这样一种启动旅程的短时间方式，他们能明确知道自己能否可以安然过冬。

“哈德逊湾式启动”技术可以让项目团队先尝试在项目的实际环境中开展某些工作。项目经理应尽量缩短这个过程。（“Hello World”程序也许就够用了。）关键是要让团队了解到，在当前项目产品所在的领域中实际工作会是怎样的状况。

如果项目经理和团队对于任何工作都没有头绪，不妨考虑用时间盒来限制“哈德逊湾式启动”。从可以在4个小时之内完成的工作开始。（这不一定是某个真正的功能。）团队有了一些成果之后，可以分析这次活动。团队会开始了解如何估算要做的工作。如果还是知道得不够，可以使用短期迭代，然后再决定接下来要做什么。

“哈德逊湾式启动”会起到多方面的帮助作用。首先，团队获得自信心，知道自己能够取得成果。在进行估算时，由于完成过一些相关工作，他们可以有更深的理解。其次，团队还能对如何组织某些任务有些概念。“哦，如果想并行开发这些任务，我们得建立新的代码分支，之后再合并。啊！这就是说我们要做阶段整合工作。相对仅在主线上开发，这会花更多时间。”

听到这样的谈话，也就说明人们开始发现并明确表达了风险，项目经理就应该把这些风险记录在“停车位”（见附录B）中，并在稍后或是在安排日程时处理。

4.2.5　短期迭代

当团队对开发环境有所了解，但是对于如何估算任务还是没什么把握时，可以使用短时间的迭代。这样，人们可以了解自己能在一到两周内完成多少工作量，这样再进行估算就可以更准确。一旦团队对环境有了深入理解后，可以在“哈德逊湾式启动”后再进行一次短期迭代。

要用时间盒限制短期迭代（不要超过两周，最好一周），再看看团队能够完成多少工作。在迭代即将结束时，团队和项目经理可以对需求、风险和其他不清楚的东西有更深入了解。

如果团队可以一起进行短期迭代和短期回顾，大家就能更清楚地知道如何安排当前项目的日程。

4.3　用低技术含量的工具安排项目日程

回到计算机的“旧石器时代”，当我开始管理项目时，我们没有计算机辅助的日程安排工具，只能用黑板、纸和流程图模板。我在黑板上画出项目的日程。黑板用起来很方便——如果哪里搞错了，就可以马上擦去并进行修正。

如果必须擦去并重新写信息，黑板可能会变得很乱。即使我跨入了“新石器时代”，开始使用白板，它有时也很难看清，因为在新的绘图下面可能仍有残存的旧信息。

进入“现代”，黄色即时贴出现了，我也开始使用它们[①]。把任务写在即时贴上很方便，再将其贴在墙上，跟团队其他成员一起讨论——有时声音很大——任务的顺序、分配或者项目的风险。而且，如果任务的摆放位置不对——因为团队发现可以通过另外的方式来组织项目——要移动这些即时贴也是很容易的。

黄色即时贴可以让整个团队参与项目日程安排的讨论。团队也会不断说明对风险的理解，并为项目经理提供掌控项目的有价值信息。

高技术含量与低技术含量

——桑迪，经验丰富的项目经理

我已经有15年管理项目的经验了。自打有了日程安排工具之后，我就开始使用了，也成为当时这类最知名工具的专家。刚开始的时候，我们想把子项目也管理起来，但是遇到一些问题，不过我找到了变通的解决方式。后来我们在跟踪项目细节的时候也遇到一点困难，但我挺善于在工具中用点小技巧解决这样的问题。在计算挣值[②]（earned value）时遇到了麻烦，但我们采取了按功能实现的方式，这也确实起到了作用（见11.2.3节）。

几年之后，我开始管理一个很大的工程，涉及6个地点大约300人。我也不糊涂，所以在一

① 有人奇怪我当时为什么不使用电脑中的日程安排工具，答案很简单：我使用的操作系统没有这方面合适的工具。因为没有，所以当然没法用啦。

② 挣值法又称为赢得值法或偏差分析法，是对项目进度和费用进行综合控制的一种有效方法。其核心是将项目在任一时间的计划指标、完成状况和资源耗费综合度量。将进度转化为货币，或人工时、工程量。其价值在于将项目的进度和费用综合度量，从而能准确描述项目的进展状态。挣值法还可以预测项目可能发生的工期滞后量和费用超支量，从而及时采取纠正措施，为项目管理和控制提供了有效手段。——译者注

开始把所有的项目经理召集到一起开计划会议。我把电脑跟投影仪连在一起，大家开始制定项目日程。每个人都在对我大喊，试图让我知道哪个任务应该归属于哪里。我是感受到一些压力，不过最后还是完成了。就在这时，停电了。

项目经理之一鲍勃说："都别动，我出去一下，马上回来，咱们再继续。"5分钟后，他回来了，手上拿着几叠黄色即时贴和几只钢笔。他解释了一下我们该如何进行日程安排，然后大家就开始往即时贴上写东西。大概过了10分钟，我们把即时贴贴在墙上，讨论每张的含义和我们之前遇到的问题。

不到一个小时，日程安排已具雏形。我们把它拍了下来，这样就不怕再出现停电或者计算机电池耗光的情况了。

会议结束时，每个项目经理都恭喜我可以这么快制定出项目日程。天！这都应该感谢鲍勃。这个日程够我们用上几个月的。后来要更新它的时候，我们又聚集在一起，做了同样的事情。

这样做的出色效果让我惊讶。我仍然使用日程安排工具，但是我总是先用低技术含量的工具来做日程安排，如果需要重新做主要的规划，我还是要用低技术含量的方式。

在开始安排项目日程时，很多项目经理喜欢使用日程安排软件。如果需要同时安排许多任务，而且项目经理认为它们的顺序不会变化，那么在项目开始时，日程安排软件也许可以发挥作用。可这样一来，就无法让整个团队人员参与了。用工具生成时间表，会绕过讨论的过程，也就很难暴露出隐含的任务依赖关系和风险。

项目经理一次只能输入一个任务，而且只有项目经理可以创建任务。日程软件一次只能展示一页信息，如果团队人员不能看到整个时间表，他们可能对整体计划没有概念。

如果没有使用波浪式规划（rolling-wave planning），当团队一起创建了初始的项目日程后，用日程安排软件应该是没有问题的。[但是会失去使用"大型可见图表"（Big Visible Chart）或"信息辐射器"（Information Radiator）带来的好处，见第11章。]可这么做会传递给团队这样的信息："负责日程安排的是我，不是你们。"

要是项目团队有时间表的所有权，他们就会对其负责到底。如果项目经理拥有所有权，这就像是在对团队展开微管理（micromanage），而不是管理他们所负责任务之间的依赖关系了。

我希望可以说服作为项目经理的你使用即时贴或是墙上的卡片。如果你不知道具体怎么做，下面是我曾在多个项目中用过的一些技巧。

4.3.1 使用即时贴安排项目日程的基本技术

将整个团队集中在一个房间，这个房间有一面很长的墙或者一块很长的白板。为每个人发一

叠黄色即时贴和一支中等粗细或更粗的黑笔。（我喜欢用3英寸×5英寸的即时贴，用签字笔在这样大小的即时贴上写字很方便，而且读起来也不费力。）如果使用顺序式、迭代式或是增量式生命周期模型，就把主要的里程碑写下来贴在墙上，这样大家可以看到项目的整体结构。让每个人把其所有的任务都写在即时贴上，每个即时贴一个任务。写完之后，他们可以将其贴在墙上。（可以参看图4-1和图4-2。）

图4-1　一个项目的黄色即时贴式日程安排

图4-2　另一个项目的黄色即时贴式日程安排

将墙的一部分作为“停车位”（见附录B），摆放大家收集到的问题和假设，项目经理应将解决这些疑惑作为日程安排工作的一部分。我使用活动挂图作为停车位，所以如果需要将问题带回办公室解决，携带起来也很方便。

现在项目经理就可以后退一步，把位置让出来了。项目团队成员开始一起讨论事件的顺序、任何前置条件、假定以及问题。

开发人员写自己任务时，会向需求分析人员、需求编写人员和测试人员提出问题，而测试人员也会有问题要问他们。在项目“启动”之前（实际上，项目已经开始了，这时只是还没有任何

人工产出，见1.6节的“在你思考之前，项目就已经开始了”），团队就已经开始像跨职能团队一样工作了。小项目看起来就像图4-1中那样。

团队成员写完所有的故事并解决完问题之后，就该项目经理上场了。在现在的日程安排中可能会有下面这些问题。

- 团队只安排了最初几周的工作。他们无法预测到几周之后的状况，因此只能安排这么多。这也没关系，因为可以使用波浪式的规划方式（见5.6节），以迭代方式产生日程安排，而且项目经理也不会希望人们提供太多没有事实依据、凭空想象的细节。更多的细节只是浪费而已。
- 项目经理可能会看到很长的任务安排序列，这要是在顺序式生命周期中还好。如果希望采取迭代式或增量式生命周期，就得问问团队：是不是有什么阻碍了他们以并行的方式工作。可参见图4-2，其中的项目与图4-1中相同，只是以更为顺序式的方式组织。
- 项目经理也可能会看到很长的并行任务安排序列。除非用顺序式生命周期组织项目，否则你不必担心。顺序式生命周期的本性决定了它无法进行并行式的开发。不过，对于非敏捷的项目生命周期来说，这预示着风险，可能造成团队的工作不同步，而且关键路径也许会以项目经理预想不到的方式延伸。

一旦团队安排好项目时间表，大家就可以开始估算每项任务将会花费的时间了。

4.3.2　使用即时贴和箭头安排项目日程

我的一个客户也是用上述的黄色即时贴安排项目日程的，不过当时间表稳定之后，他们会在即时贴之间画上箭头，这会有多种帮助作用。首先，如果一张即时贴掉下来了，他们会知道再放回什么位置。其次，用即时贴做完日程安排后，可以将其结果录入到日程安排软件中。项目协调人会把每张即时贴转换成工具中的任务，箭头就可以辅助跟踪依赖关系。

4.3.3　为每一个职能组使用即时贴安排项目日程

项目经理如果只能使用阶段–关卡式的日程安排，也不能创建跨职能的团队来按照功能进行实现，那你可能需要用一种日程安排方式来说服管理层：还有其他的选择。我曾经使用即时贴式日程安排，通过让管理层察看每周的时间计划，让他们知道按职能组织的团队会拖项目后腿。

在一个大白板或是粘在墙上的白纸上画几条竖线，每条表示一周。使用不同颜色的即时贴，用其表示处于不同职能组织的不同人的工作状况。

别忘了把项目的结束时间表示出来。按职能组织的项目在结束时会很难受，因为管理层认为开发人员那时可以进入另一个项目的开发了，在项目最需要他们的时候——需要修复缺陷时，他们却很难全心投入。

4.3.4　按功能使用即时贴安排项目日程

最近，我开始使用即时贴式的日程安排，查看各个功能之间的结合情况。如果管理的项目很复杂，在集成项目时会发现很多依赖关系，这时不妨通过即时贴规划一个迭代要完成的工作任务，这会有很大帮助。

把每个可交付物用一张即时贴写好。有时，一个功能会有多个临时可交付物。把即时贴贴在墙上。当团队需要将可交付物的代码集成到代码库中时，由他们自己去组织。让团队写明严格要求的截止日期，“如果你到时候不能交付这个功能，那么在这个迭代结束时，我们就完不成任务了。”

尤其是短期迭代的情况，项目经理不必将即时贴上的内容生成甘特图。如果使用增量式生命周期模型，对于时间较长的项目，项目经理可能要把即时贴用胶带粘起来，或者使用甘特图管理依赖关系。

4.3.5　使用即时贴安排项目日程的好处

使用即时贴安排日程，大家可能看不到那张美丽的、展示项目关键路径的甘特图。这没关系，因为软件项目的关键路径是通过任务、人，有时是设备来体现的。而且，由于受大家每天所完成工作的影响，我敢说你的关键路径每天都在变化。即使不是每天变化，每周也一定会发生改变。如果没有甘特图展示关键路径，项目团队就得考虑这个问题。每个人都要更有意识地去思考关键路径，这也有助于形成大家对其的责任意识。

此外，黄色即时贴是不会展示结束日期的，因为项目经理绝对不能去估算唯一的结束日期［DL03］。但是日程安排工具软件会计算结束日期（它给出的日期可能是最早的结束日期，项目经理也无法证明届时项目不可能完成），而人们，特别是公司高层，会相信这个结束日期。

如果项目经理负责一个多地点的项目，仍然可以使用即时贴安排项目。要是每个团队负责一个完整的可交付物（一系列开发完成、测试通过的功能，见12.3.1节），这些团队就会自己做每天的日程安排。项目经理应该把各个团队的带头人或项目负责人召集到一起，确保他们理解：谁负责在何时将什么交付给谁。此时要解决的是主要里程碑，可以使用视频会议或网络会议系统，并使用类似于即时贴的方式来安排项目日程。

4.3.6 基于可交付物的规划

基于可交付物的规划（deliverable-based planning）也可以用黄色即时贴来实现。人们会思考必须要为项目剩下的工作交付哪些东西，他们的里程碑会基于可交付物，而不是基于项目进展阶段的结束。

基于阶段的规划或是基于功能的规划，会假定特定职能的团队负责项目的一部分。有人说自己完成了工作，项目经理就可以假定项目已经完成一个阶段了。如果所管理项目的里程碑类似“需求冻结”或“代码冻结”，那这个项目就是基于阶段进行规划的。

问题在于，即使大家都努力完成自己负责的可交付物，所谓的“冻结”也是很少冻结的，而“完成”的也基本上无法完成。这样得到的里程碑也是“泥泞不堪”的。要避免这样的情况，在规划里程碑时，要把其之前的任务在此告一段落。如果你知道有多个不同领域的需求，里程碑“需求冻结”可以改成“需求1完成编写和复查”、“需求2完成编写和复查”或“需求3完成编写和复查”，等等，直到所有的需求进度要求都已说明。

在任何生命周期中，都可以使用基于可交付物的规划。特别是对于顺序式的生命周期，使用基于可交付物的规划可以尽早获取关于项目进度的反馈。如果不能冻结需求，你怎么能知道可以完成之后的里程碑呢？

提示：延迟的项目无法弥补时间进度，一定会延迟

在项目开始时，如果项目经理认识到团队没有按进度进行，那他就得想点别的办法了。延迟的项目是无法弥补时间进度的，它们会变得越来越迟……

如果认为项目可以弥补时间，你就会发现自己陷入了6.15节所说的日程安排游戏之中。

铭记在心

- 用低技术含量的工具开始安排项目日程。如果真的需要相关软件工具，过后再转换数据。要注意不使用“大型可见图表”或“信息辐射器”所带来的成本。
- 按可交付物安排日程，而不是按功能。
- 要有以迭代的方式安排日程的准备。一次完成的项目时间表，其作用根本无法对得起花在上面的时间。

第 5 章

估 算 工 作

5

项目日程已经安排妥当，接下来该估计每个任务要用多久才能完成了。项目经理大可不必“姑妄猜之”，其实还是有很多种选择的。选择方法时，它所提供的估算结果应该是你最需要的，而不是最准确的。

5.1　实用的项目估算方式

我成功使用过的估算技术有以下几种：历史数据、Delphi、宽带Delphi、相关排名和大小，并在估算前使用试探性开发（spike）收集相关数据。任何通过计数或是计算的技术，我都没有成功使用过。关于计数和计算的技术，可以查看McConnell的相关著作［McC06］。

5.1.1　通过对比历史数据进行估算

如果你所管理的项目的产出是某一产品的后续发布版本，那这个项目的持续时间也许是可以估算的。“嗯，我们8个人、6个月完成了上一个版本，这次看起来规模差不多，所以我的初步估算是一样的。”要记得项目是非线性的。如果这个项目的规模看起来比前一个只大一点点，历史数据对比可能就起不了多大作用了。将历史数据与Delphi方式或是宽带Delphi方式结合起来，可以收到更好的效果。

5.1.2　通过Delphi和宽带Delphi方式进行估算

通过Delphi方式进行估算，项目经理会把团队召集在一个房间中，并就项目有关事宜向他们进行阐述。大家提出各种问题，每个人都写下来自己的任务列表和时间估算，同时注明自己对项目的预设条件［Wie00］。

接下来，团队收集任务列表并进行复查，看看哪些任务可以同时进行。项目经理再把估算时间加在一起，得到项目的整体估算。

如果项目经理不能接触到从事实际工作的人——真正的项目团队，此时不妨使用宽带 Delphi 方式。一小组专家会暂时替代项目团队，产生任务列表和估算。在完成估算后，他们会提交任务列表、预设条件和风险。

使用这两种Delphi预测方式，其效果都要好于项目经理自己进行估算。但是，它们都有同样的问题——每个人负责估算项目的一个部分，而且通常是按架构进行切分的。使用这两种Delphi方式，你可能会遇到我曾碰到过的问题——估算过低。应对这个问题的唯一方式，就是在估算时将任务大小与持续时间分开考虑（见5.1.8节）。

5.1.3　何时不应相信团队的估算

不是每个人都善于估算。有些人会过于乐观［Bro95］，对于任何任务都会少估算50%的工作量。有些人会过于悲观，为每个任务添加缓冲的时间。有些人在估算不超过3～4天的小任务时没有问题，但是对于超过一周的任务，却很难估算准确。项目经理该怎么办？

首先，要了解你的团队。要想清楚，对每个人在项目中的估算提供多少反馈。你没有必要在项目第一周时就解决所有估算问题。

接下来，要去掉每个任务的任何额外缓冲时间。询问大家在估算时是否考虑了缓冲时间。跟大家说明，你不是要减少任务的完成时间，而是要确保能够得到最准确的估算。

如果使用顺序式或迭代式生命周期，在估算时可以考虑使用约束理论（TOC）［Gol04］。依据约束理论，每个人会提供一个他认为合理的任务估算。项目经理取其时间长度的一半，在甘特图中将该任务标识为对应的时间长度，再取估算时间的25%放入缓冲区中［Gol97］。如果关键路径中的任务需要更多时间，可以从缓冲区中取出时间加在这个任务上。再继续管理缓冲区。当团队完成所有的任务之后，通过计算缓冲区中的时间，就可以知道之前估算的整体效果如何。

在增量式生命周期或是敏捷生命周期中，项目经理要收集已完成工作所花费的时间，并将其与估算时间对比，随项目推进不断学习和改进估算技巧。

总地来说，不要为任务的估算添加过多松散的时间。要提供一个估算信心百分比（见5.1.5节）、一个日期范围（见5.1.6节），甚至可以考虑提供三个日期：最佳状况、最有可能、“墨菲”日期（见5.1.7节）。

有人会问：那什么时候才能向估算中加入松散时间呢？要将此作为最后一招。跟团队成员一起定义出一个“估算质量因子”（EQF，Estimation Quality Factor，见11.2节），用其来监控他们的任务。使用带有时间盒限制的迭代，这样人们就可以少一些估算工作，而且估算的粒度也会更小。为估算提供反馈，它们既可能来自项目经理，也可能来自项目团队。但是如果拉长估算的话，项目经理就可能会引入“帕金森法则”或是引发“学生症状”（Student Syndrome），并因此担上风

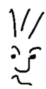
险。如果人们可以一直无忧无虑地忽略他们当初所做的估算，那他们就永远不会学到如何改进自己的估算技能。

5.1.4　小心顺序式生命周期的估算陷阱

对于使用顺序式生命周期的项目经理来说，他们总要在一开始就负责估算并安排整个项目的时间表。这会花去他们（以及参与进来的项目团队成员）几周的时间，因为他们要对需求和系统架构有足够的理解，才能产生合理的估算。基于组织对该项目的陌生程度，他们的估算会发生偏差。我曾见过从100%到400%不等的偏差幅度。

从一开始就估算整个项目，这是个陷阱。如果必须使用顺序式生命周期，是很难对项目做出准确估算的。要想做好，可以让团队先动手做一点开发工作，测量出完成这些工作需要多长时间，再看看项目中包括多少类似工作量的工作，并通过迭代方式得到估算。

特别是对于使用顺序式生命周期的项目经理来说，要利用估算信心百分比（见5.1.5节）和日期范围（见5.1.6节），这样大家就可以知道估算的不确定性有多大了。



小乔爱问……

什么时候才能向估算中加入松散时间呢？

人们很容易乐观，并过少估算完成任务需要的时间。要想解决这个问题，看起来必须要拉长这些估算。但是问题在于“帕金森法则”：工作会占满为其分配的时间。

假设你的首席开发人员很乐观，他对任务的估算是16小时。可是你知道，这个任务得用20～30个小时才能完成。作为项目经理，你该怎么做？

首先，要帮助开发人员在估算时分开考虑规模与持续时间。可以参见5.1.8节。如果他的任务量很大，让他把任务切分成“小石子”（见附录B）。他可能会说：“噢，这个任务有16个小时，两天之多啊。”如果他能保证每天工作8小时，那没问题。但实际上他被干扰的次数比项目中其他任何人所受的干扰都多，所以不能保证每天8小时的技术工作时间。根据环境的不同，我估计首席开发人员每天花在技术上的时间会有4~5个小时。16个小时的任务可能就会耗费他3～4天的时间。

如果“小石子”的做法不可行，试一下“试探性开发”（见5.1.10节）。试探性开发有助于大家看清任务要用多久才能完成。

让低估任务完成时间的人使用三个时间（见5.1.7节）作为提供估算的基础。这样开发人员即使得不到项目经理的反馈，他也会心里有底。

与习惯性低估任务完成时间的人交谈，帮他们认识到自己实际能够完成的工作量要比他们估算的少。为了保护项目，考虑使用缓冲或是采取迭代方式开发。

因为迭代的时间很短，每个人都能在估算后几周内就得到关于自己估算的反馈。团队成员可以互相纠正：“上个迭代中，你说那个功能的规模是3点，可实际上是8点。这个功能跟那个很像，能告诉我你为什么认为它还是3点吗？

提示：项目日程并非一成不变

日程安排是由整个项目团队共同制订完成的，所以每个人都对日程有信心。可是“天有不测风云”，总会发生点儿意外。

别担心。项目日程是团队当前对于项目工作会如何展开的估计。随着时间的推移，意外情况发生了，通常在日程“完成”一到两天之后，最初的日程就变成明日黄花了。这就是我为什么只做足够的时间计划，而且会使用波浪式规划，这样我就可以随着环境变化很容易地更新日程安排。

5.1.5　使用自信心范围进行估算

你对自己的估算有多大把握？项目刚开始，我一点儿把握都没有。我只知道项目不会完全按照我们制订的日程推进。有些东西是会变的，它们会改变估算。不要使用对结束日期的单点估算，尝试使用自信心范围吧。

可以看看图5-1，这就是自信心范围图。可能性的范围就是自信心的水平。在项目刚开始时，项目可以完成的可能性为0。

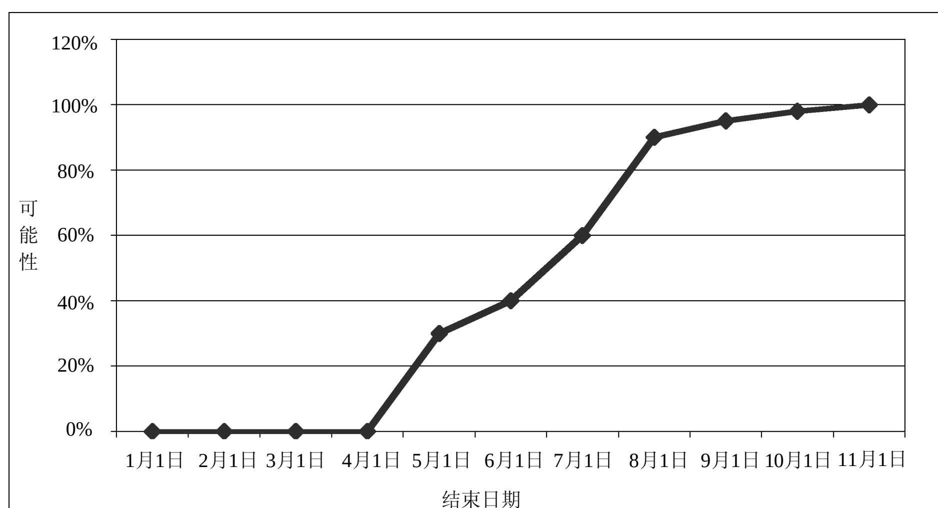


图5-1　自信心范围图

在60%这一点上，7月1日是项目可能完成的最早日期。注意，这个最早完成日期也仅仅是可能而已。

80%可能性的日期在7月中旬。从项目刚开始时看，这个日期完成项目的可能性更大了。注意趋势线的斜率变得更小了。几乎不可能在项目启动时就得到确定的发布日期，这也是为什么9

月、10月、11月都在90%～100%的范围之内。

类似图表可以让你与出资人进行如下的对话：“要想在6月15日发布，我只有50%的信心。我知道你想在那时候要，但是我无法证明项目在那之前可以完成。”接下来，你就可以解释为什么对这个日期没有信心。

要是项目经理用顺序式生命周期组织项目，可以使用图5-2中所示的不确定性锥形图。它可以解释为什么在项目启动时的估算不准确，以及何时可以得到更准确的估算。当项目能够在不同时间满足不同阶段的里程碑时，根据不确定性锥形图得到的重新估算也就越来越准确。

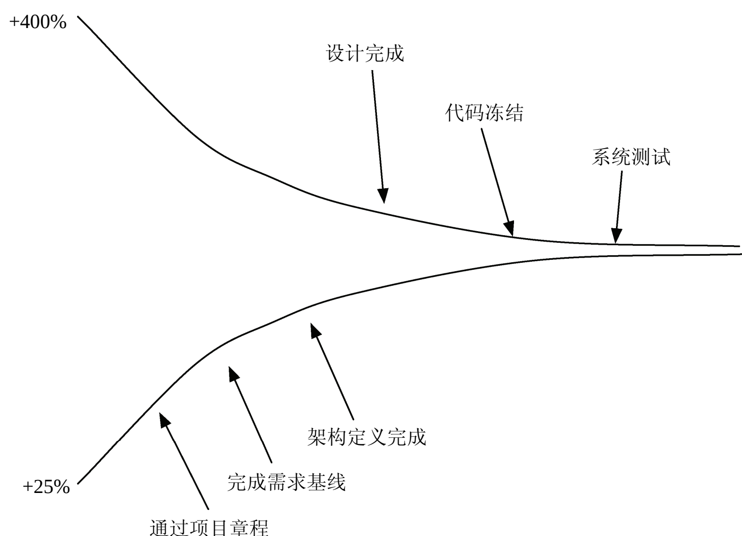


图5-2　不确定性锥形图

如果你同时得到了早期的里程碑，比如需求冻结、设计冻结和代码冻结，[①]只有在代码冻结时才能使用锥形图重新评估，在此之前都是无法重新估算的。

如果使用迭代或增量式生命周期，项目经理也可使用上面的锥形图。生命周期的增量划分越细，再次估算也就可以越快进行，而且准确度越高，因为这时有真正的数据支撑后面的估算。

如果使用敏捷生命周期，就不必用锥形图了。项目经理只要测量团队开发速度和需求变化率即可，这样可以预测发布日期。

① 也许有人会疑惑为什么“需求冻结、设计冻结和代码冻结”都被称为早期里程碑。作者是这样解释的：“很多使用阶段式或顺序式的项目，在代码真正冻结之前，‘代码冻结’这个里程碑之前的各个阶段实际上都无法完成。因此，虽然‘需求冻结’和‘设计冻结’被看作早期里程碑，但是却只能在后面的里程碑真正达成时同时完成。从这个意义上来说，‘代码冻结’也可以算作早期里程碑。”——译者注

5.1.6　使用日期范围进行估算

如果有人要求项目经理进行总体估算，而且团队中没有其他人可以施以援手，不妨用日期范围的方式，并解释何时可以修订估算。下面就是我使用日期范围的例子。“好吧，基于你在三分钟时间里告诉我的有关这个项目的信息，看起来我们能够在第三季度交付一些东西。今天是1月10日，我得用一周时间找齐人员。让我做一些迭代，再跟团队一起规划一下，我会在2月1日给你一个更好的估算时间。”

团队启动之后，我们也大概知道了可以在两周迭代内所完成的工作量，这就有了更多信息。我是这么跟经理说的：“我们觉得第三季度这个时间太乐观了。不过在10月到12月初这个时间段内完成应该是可行的。”到了4月份，我跟经理这么汇报：“现在看来， 11月底到12月初完成应该问题不大。过几个月我给你更准确的消息。”到了7月，我的汇报是：“仍然应该可以在11月15日到12月1日完成。你什么时候需要更准确的时间？”

使用日期范围，可以让项目不至于受过早承诺之苦，不必在一个不可能的日期之前完成。不过，要是你说“10月到12月”，而经理只听到了“10月”，那你最好还是用自信心水平来解释。或者，项目经理可以使用敏捷生命周期，并且一直用待办事项列表进行管理，这样就能尽早满足经理对于交付日期的要求。

5.1.7　使用三个日期：最佳状况、最有可能、“墨菲”日期

有些高层不喜欢从项目经理那里听到一个不断变化的日期。这时，项目经理可以使用“三个日期”［DL03］技术：最佳状况、最有可能、“墨菲”日期。

最佳状况日期，就是项目最早的可能完成日期。在生成甘特图时，最佳状况日期就是工具提供的日期。即使已经有缓冲时间或者准备好了某些应急措施，项目经理还是可以确定：由于某些原因，这个日期是不可能的。可你不能确定地说项目在此日期时根本不可能完成。这也是为什么说它是最佳状况日期。

当项目经理将一些经验系数（fudge factor）、缓冲时间纳入考虑之中，或是将估算加以延伸之后，这样得到的日期就是最有可能日期。项目经理对这个日期更有信心，但也只有80-90%的把握。

如果你认为“墨菲法则”[①]会一再地在自己的项目中发生，由此产生的日期就是“墨菲”日期。在大家放大假的前一个星期，该死的暴风雪袭击了佛罗里达，台风摧毁了东南亚的数据中心，

① “墨菲法则”是说：如果某事可能会出错，它就一定会在最不应该出错的时候出错。

或是电力公司的一个变压器发了生故障，这些问题在几天到一周之内都是可以解决的。彻头彻尾的灾难，像丢失了所有的源代码，或是项目团队在同一天全部离职这样极端的事情，所造成的日期并不是“墨菲日期”。

要得到“墨菲”日期，可以在最有可能日期之上加入一些经验系数。只有项目经理知道应该加入多大的经验系数。

提示：估算需要准确性，而不是精确性

上面提及这些技术，通过展示估算不精确的方面来强调估算的准确性。在项目早期，是不可能知道项目准确发布日期的。除非项目团队可以自主定义要发布的功能和质量要求。

精确性就是测量的精密程度，可以用小数[1]表示。准确性就是看看估算离实际情况有多大差距。项目经理关心日程的准确性，关心对任务持续时间或安排的预测离实际情况有多远，而不是某项任务或整个项目会在某一天的几点完成或结束。

不要担心估算的精确性，要更注意它有多准确。

5.1.8　在估算时分开考虑任务的大小与持续时间

人们不善于估算自己从未做过的任务。谁又可能做得到呢？要想解决习惯性低估工作量的问题，就得把任务的大小估算和持续时间分开考虑。

任务的大小就是它的规模。通过粗略估计，可以判定一个任务是小型的还是中型的或是大型的。但对于绝大多数项目来说，粗略估计还不够。项目经理需要更细粒度的估算，而且还要把粗略的估算转化为任务的持续时间。

科恩在《敏捷估算与规划》［Coh06］一书中建议使用斐波那契数列来粗略估算任务。任务的大小可能用1、2、3、5、8、13、21、34、55、89等等表示。如果项目团队过去的估算都比较准确，这个数列的数字用到21就可以了，往上用40、60、80和100就行。有时在正式估算某个任务之前，团队还得安排试探性开发（见附录B），以决定任务的真正大小。（对于不善于估算大任务的团队来说，要让他们使用21、40和100，而不用60，80等中间数字。再通过试探性开发，把大任务拆分成更小的部分。）

一旦使用斐波那契数列（或是其他的方式）得到了各个任务相对的大小，取出所有估算值为“2”的任务。看看是不是这些任务的大小都差不多？如果是，现在再估算这些大小为2的任务会持续多少时间。假如认为需要10人时（person-hours），将类似任务的持续时间除以2，就能得到大小为1的任务的持续时间，即大小为1的任务会用5个小时。问问自己这样做是不是有意义。如

---

① 请查看：http://www.ayeconference.com/Articles/Estimateprecisionaccuracy.html。

果是，你就有了基本的因子，可以用它乘以任务的相对大小，得到其他任务的持续时间。

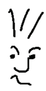
要是团队对于相对大型任务的估算不太有把握，项目经理就会知道持续时间的不确定性有多少，并就此进行日程安排。（任务越大，不确定性越大。）而且，如果团队把所有任务估算为“13”甚至更大，就说明团队还没有将任务拆成更小的部分，这对于整体的日程安排来说也是一个很大的风险。

你可能注意到，我建议使用人时估算，而不是理想日。因为每个人对理想日的理解不同。一个花费大部分时间来指导其他人的资深开发人员，每天的工作成果甚至可能不如初级开发人员完成的多，因为初级开发人员理解自己负责的部分系统，而且外界干扰要少得多。

5.1.9　规划扑克

项目经理想让所有的团队成员都能参与到估算活动中，可大家从来都不知道估算的效果怎么样。面对这种情况？项目经理该如何开始？用规划扑克［Coh06］吧。

团队中每个人都会对待办事项中功能的相对大小做出判断（见16.6.1节）。你可能会说：“OK，这个功能是让排序变得更安全。”大家想了几秒钟，然后每个人会估算这个任务的大小，展示一张写有自己估算数字的纸片。如果团队由6个人组成，每个人都认为某个任务的大小是5点，那么项目经理可以放心地把“5”写下来，表示这个功能的大小。如果有人估出13点，可别随便计算出一个平均值就算完了。要问问他出于何种考虑。你可能听到：“之前我做过类似的功能，我们发现了很多过去没有想到的异常情况。”之后再让大家估计一次，看看是不是有了一致的意见。（要让大家都接受一个数字，而不是达成完美的协议。）如果很多人觉得它是8点，而其余认为是13点，那就再问问大家能够接受哪个结果。如果不能达成一致，不妨安排一次试探性开发任务，完成之后再重新估算。



小乔爱问……

估算时应该用“人时”还是“人日”？

很多人喜欢用人日，也就是理想日进行估算，而不是用人时。

但是一个人很难在一个工作日内完成8个小时的工作。我所见过的最好状况，也就只能在一天内完成6个小时的技术工作。有些同事跟我说，由于各种会议及其他干扰，他们每天最多也就能干4个小时的活——这还是最好的状况。坐下来看看你的项目团队和环境。也许你的团队每天最多只能完成2～3小时的开发工作。

当你和团队以理想日进行估算时，所假定的每天工作时间会超出实际情况，从而导致估算发生问题。类似状况十有八九，经常发生。

规划扑克集Delphi和相对大小估算的好处于一身。整个团队都可以参与进来，而且团队可以很快地完成待办事项相对大小的估算。

5.1.10　在估算前用试探性开发收集数据

有时，我们知道一个任务很庞大，但是搞不清楚它到底有多大。项目经理也不知该如何估算。“大”不是一个足够好的估算。此时，可以尝试试探性开发（见附录B）。

看这个例子。假定你所在的团队负责提升产品某部分的性能。团队中没有人知道具体该怎么做，所以谁也不知道会用时多久。团队可以用一个短期的时间盒——也就一天左右——来看看完成实际任务需要多久。在时间盒结束时，大家对于这个提升性能的初步任务就应该有所了解了。要是对性能提升还了解得不够，还可以再进行一次试探性开发。

试探性开发可能更短。如果团队中有人想知道某个任务是不是要花费60小时，他可以先用2~3小时（也许同其他人一起）重新定义整个任务，将其拆分成一个一个的小块，每个小块耗时4~6小时。

如果团队成员不习惯用小块任务的方式来思考，试探性开发也许有助于他们试着将任务拆分成最小的部分。

5.1.11　让估算更容易的提示

下面这些方式可以让估算和报告估算变得更容易。

- 记住，估算只是一个大概值——一个猜测。猜测的范围越大，可能产生的错误也就越多。在对项目完成“日期”进行估算时，要提供一个日期范围，让听众知道你的估算只是猜测。
- 很多软件开发人员都是很乐观的。在学校时接受的训练就是要让他们变得乐观，因为那时每个项目都会在一个学期内完成（还要花费许多个通宵）。这些训练会一直停留在他们脑海中，直到学会估算小的功能，并接收对于估算的反馈。
- 完成一项任务总是要花费比预计更多的时间。
- 估算小块的工作更容易。
- 决定估算采取的时间单位是用“人日”还是“人时”。我推荐用人时。
- 项目经理和团队需要练习估算并收集反馈。没有反馈的估算只是“写下来的估算”[1]——它让你感觉良好，但是不能产生长期的结果，最终只能被证明毫无价值。

① 感谢基思 • 瑞（Keith Ray）提供这个词汇。

- 做好反复估算的准备。在项目进行到一半的时候意识到估算过于乐观并不晚，花点时间重新估算，重新安排剩下的工作。已经延迟的项目是无法再赶上进度的，它们一定会延迟交付。即使项目看起来不会晚，也应该花时间重新做估算。
- 如果项目经理必须遵循某个截止日期，那就干脆什么都别估算了。把所有的功能进行排序，并按优先级进行开发。这种状况下，我强力推荐使用敏捷生命周期，这样就可以快速地实现功能，并马上得到反馈。如果不能采纳敏捷生命周期，考虑使用增量式生命周期吧，随着项目进展按照功能逐个进行实现。
- 如果项目被限制得很死，就要用时间盒把各个阶段和任务包围起来。
- 如果任务过大（包含很多技术风险），不容易估算好，可以先用试探性开发。

如果截止时间定死了，就别浪费时间进行估算了

CIO丹很清楚这一点："这个项目必须在4月11日之前完成。"4月12日是重要的演示日期，而且丹想让大家看看这个项目做得多么出色。项目经理塞茜尔已经惯于管理不受时间约束的项目了，这些项目只要在发布前具备一些特定功能即可。通常，塞茜尔会通过估算原型和反馈时间来开始项目。她很明白，之前的方式在这个项目是无法发挥作用了。

塞茜尔决定团队仍然按照逐个功能进行实现，但是不需要估算每个功能的开发要用多久。她所要做的就是制订出功能的开发顺序，这样即使在演示之前不能做完所有的功能，也能保证交付最重要的部分。

塞茜尔和团队大体估计了每个功能，但是没有试图得到更详细的估算。他们没有在4月11日之前完成丹需要的所有功能，但是已经很接近了。而且，他们交付的都是已经全部完成的功能——而不是一大堆半成品。

塞茜尔和团队没有浪费时间做多于实际需要的估算工作。他们在一个小时内就完成了整个功能列表的估计。塞茜尔记录下了完成每个功能实际用去的时间，这样她就可以预测团队在4月11日之前能完成哪些功能了。她没有用太多时间做无谓估算，因为已经知道截止日期了。

5.2　用里程碑切分项目

如果要估算的大项目不是几个人、几周时间就可以完成的，不妨定义一些里程碑，这样项目经理和团队就可以对要估算的东西有所了解。用可交付物作为里程碑，不要用与功能相关的活动。

如果使用顺序式生命周期，要记得每个阶段的结束都能把这个阶段内的交付物包括在内。

提示：使用基于可交付物的规划来安排任务

基于可交付物的规划就意味着，项目经理和团队在制订日程时要依据可交付物而不是与功能相关的活动。如果开发的系统需要架构原型，那就得有这些基于可交付物的里程碑：为架构制订三种可选方案、审查这些可选方案、选择一种方案用原型实现、完成架构原型（作为最后的里程碑）。另一种规划可以是：实现功能1、实现功能2、实现功能3、评估当前架构、选定项目架构、完成架构原型。这两种规划都有一个收拢式的里程碑——完成架构原型，它基于项目团队的可交付物。

一个阶段或任务何时才算"完成"？没有可交付物用于帮助理解如何完成某个里程碑，团队又该怎么交付名为"完成架构原型"的里程碑呢？上述任何方案都可以帮助团队实现"完成架构原型"的里程碑，而且每个方案的完成方式不同。（第一种方案是供迭代生命周期使用的。第二种方案供增量式生命周期使用。在决定"最终"架构之前，使用敏捷生命周期也可以实现多个功能。）

要使用低技术含量的方式做项目日程安排，特别是在管理时间紧张的项目时。项目受时间约束越紧，团队制订日程时间安排的要求越急迫，而且要保证每个人可以接受这个时间表，并且努力去达成它。

花些时间和团队成员在会议室中安排项目日程，可以用黄色即时贴，要让大家看到潜在的约束条件。

提示：将里程碑（或迭代）的结束安排在一周之中的某天

大家都愿意将主要（和次要）里程碑的结束日期安排到周五。这样一来，每个人在回家的时候，就能知道他们已经完成了一部分主要的工作。不过，生活很少能够这么理想。

如果组织项目的方式越接近顺序式生命周期，那么将周一作为开始、周五作为结束就越危险。周五结束，意味着在周一之前，没有人会检查已经完成的工作。人们就有可能在周末疯狂加班，以满足流程对周五的日程要求。而且，如果使用的生命周期越接近顺序式，项目经理（或团队）就越难以搞清楚工作是不是真地做完了，除非在项目即将结束时进入测试阶段。

使用迭代也有类似的问题。如果迭代预计在周五下午结束，但是产品演示却安排在周一上午，团队们就会想："哦，我们可以在周末再完成／修复这个功能。"特别是转向使用敏捷开发之后，团队的估算可能不大准确，也就是说他们在之前的迭代中可能有未完成的工作。项目经理想知道的，是团队在合理的时间之内能够完成的工作量，而不是要看算上加班能做多少。如果在周五结束迭代，就等于无意识之间允许团队人员用周末的时间加班。

当项目经理将里程碑或是迭代的结束安排在周中时，有哪些工作尚未完成就很明显了（项目经理也希望看到这一点）。你可以调整项目，因为你知道哪些完成了，哪些还没有完成。可如

果不知道哪些工作还在进行中，你所能做的调整选择就很少了。

可以选择周二或是周三作为主要的里程碑或是迭代的开始/结束的日期。这样项目经理可以看到真正的进展（或是未完成的工作），你可以减少加班，而且能够调整项目，不会被项目结束时可能发生的灾难所吓到。

5.3　你们能够不做哪些事情

项目团队经常会想自己能完成多少工作。他们认为项目应该围绕这样的思维方式开发："这个项目我们能做哪些事情？"其实可以换一种思维方式，想想"我们能够不做哪些事情？"

"能做哪些事情"的思维方式基于下列假设。

- 人不是稀缺资源。我们可以马上让他们投入工作，把精力都放到项目中来。
- 日程安排其实无关紧要。
- 开发的成本不是驱动因素。

"能够不做哪些事情"的想法基于下列假设。

- 对需求的理解是稀缺资源。理解特定的需求，并知道其对客户的价值所在，并全心全意交付这些需求，这才是我们应该做的。
- 日程安排至关重要，我们没有时间再做一次，也不能留下技术债务（见附录B）。
- 项目成本非常重要，我们应该把它们管理起来。

项目经理（和他们的资深经理们）经常会说，能够以"不做哪些事情"的方式思考是很重要的特质，但他们却以"能做哪些事情"的方式进行管理。下一次要是管理层问起来"能做哪些事情"，你不妨问问他"能够不做哪些事情"。如果都得做，那你就帮助大家澄清了他们的假设。

5.4　身背多个项目时的估算

项目团队的一些成员不只负责这个项目，他们还有其他项目的任务。项目经理要如何估算他们的工作会耗时多久呢？

不要估算。也不能估算。想都别想。

团队中有身背多个项目任务的成员，这个项目必然会延迟。而且项目经理都无法预测会晚多久，因为你不知道这些人能在你的项目上投入多少时间，甚至很难知道这些人在项目需要他们的时候，能否真正在场。

如果有这样的人，项目经理是无法估算工作与安排日程的。（多任务会浪费20%到90%的时间，见［RG05］和［Wei92］。）从我的经历来看，身背多个项目的任务[1]是造成项目延迟、无法按要求交付、无法按质量交付的最大原因。

项目经理要跟出资人讨论，跟他们解释说："如果不能在我需要他们的时候提供这些人，那我就不能交付你想要的东西，不能满足你对时间和质量的要求。咱们看看有哪些东西是你不需要的吧。"（见5.3节。）

如果出资人不肯让步（他们希望在没有付出的情况下，能够马上得到完美的功能），那项目经理就得说"不"了——当然是以组织可以接受的方式。可以参考16.7.3节。

5.5　主动安排人们进行多任务

有时，项目需要的有些人不必专职投入。比如一个DBA或是GUI设计人员，项目经理需要他，但不是总需要。或者，项目经理管理的多个小项目需要开发人员和测试人员，但是这些项目都不是会占用所有时间的项目。你该怎么办？

项目经理可以主动安排人们进行多个任务。对于项目中只是一时需要的人员，可以让他们以周为单位来切换项目。或者，也许团队成员可以进行结对。项目经理让一对开发人员参与两个项目。这样以来，人们就可以在为期一周的迭代中工作了，从周一开始参加一个迭代，周五结束这个迭代。没错，这违背了将里程碑安排在周中的想法[见提示：将里程碑（或迭代）的结束安排在一周之中的某天]。只有在每个人都将周末作为休息和为变更工作内容做准备的时间，这才能起作用。

项目经理会因此而付出一些代价——背负多项任务的人要完成工作会花更长的时间。这也许比雇佣更多的人节省成本。但是要确保他们一周只完成同一个项目的工作，而不是去做两个项目的事情。

5.6　使用波浪式规划

不要在一开始的时候就想着把整个项目都规划出来。这样会出错，而且也是浪费时间，本来可以用这些时间去掉项目的障碍，让团队找到自己的节奏。使用**波浪式规划**（见附录B）产生初始日程，在项目进行中调整日程，并在必要时重新规划各种活动。

---

① 造成的成本，可见http://www.umlch.edu/~bcalab/multitasking.html。

如果项目经理习惯于安排整个项目的日程，用波浪式规划也许会觉得有点不适应。你不会产生完整的甘特图，也无从得知三个月后具体要做什么工作。不过老实说，对于那么远的事情，你能安排该多好呢？我做得不好，尤其是对于项目中的事情来说。里程碑越遥远，对于到底如何到达那里就知道得越少。因为无论团队的估算有多好，总是会有突发事件阻止他们按原计划完成项目。

波浪式的规划就是一个持续不断而且很详细的日程安排，只覆盖几周的时间。完成了一周详细工作安排之后，可以再继续安排一周详细的工作。使用四周的波浪式规划，四个星期的详细工作安排就在其中，不多也不少。项目经理也不会浪费时间去规划了解不充分的事情。

我总是喜欢使用为期两周的迭代来推进项目。要是不能这么做，我就会选择使用四周的波浪式规划。由于不能使用两个星期的时间盒来安排项目节奏，如果波浪式规划时间少于四周，那我就无法得到足够的信息来预见风险。安排多于四周的计划会很容易犯错，并偏离安排的工作，所以我根本不考虑这么做。项目经理可能会觉得，自己的预测能力到不了四周那么久，这没关系，在你力所能及的范围做详细的工作安排，不要试图超出这个范围。

如果你从未尝试过波浪式规划，可以这么做。找一个足够大的房间，可以在墙上或是白板上安排和组织工作。在黄色即时贴上写下里程碑并排列好，从左至右，因为时间是从左至右进行的。接下来让项目团队成员进入这个房间。

向大家解释，这次不再一次性把项目的详细日程安排出来了。项目经理已经标出了要在什么时间完成哪些主要的里程碑，就在墙上的黄色即时贴上。

提出有关第一个里程碑的问题：“我们要怎么做才能到达这个里程碑？”接下来让大家在即时贴上写下自己的任务和任务之间的关联，每个任务一张即时贴。

让大家以“小石子”（见附录 B）的方式进行规划。由于项目经理不会把“小石子”这样粒度的任务分配到人，所以每个成员要详细理解自己的任务，并定义出完成这些任务需要的“小石子”。

如果项目团队不能以“小石子”的方式规划，就要让他们帮你理解他们的流程。不是每个人都能以“小石子”的方式思考，他们需要时间来学习如何分拆任务。

如果必须用甘特图，就把每张即时贴上的内容复制到你最喜爱的日程安排工具软件中吧。每周在与团队成员见面时，可以让他们告诉你接下来要做哪一组任务，你就可以更新日程安排了。如果人们自己不能独立完成任务的依赖分析，就再把大家召集到一起。

只要在项目进行时牢记每个里程碑，项目经理就会发现日程越来越容易维护了，花在上面的

时间也越来越少。你也可以用更多的时间跟团队在一起，看看他们的进展，并帮助他们移除障碍。

波浪式规划并非灵丹妙药，要想理解项目的真正状况并规划如何到达下一个里程碑，仅用它是不够的，但却是一个好的开始。

5.7　决定迭代的持续时间

项目经理决定使用时间盒限制的迭代来管理项目。但是迭代的持续时间应该是多久呢？



小乔爱问……

如何在短迭代中开发大任务？

复杂的项目会有大任务，应该将它们拆分成团队可以在四周或更短时间内完成的小任务。这样做有点难度，但还是值得的。

当有团队成员告诉我一个任务要好几天才能完成时，我会询问他进入代码库的可交付物是什么。我想知道这位成员是否先做设计——要用多久——或者是否进行原型化。如果埃里克打算先做设计，我就问他如何知道设计何时完成；如果他在做原型化的工作，我就问他打算如何评估原型。

人们常会告诉我，手上的主要工作就是编码。这很容易——他们会把部分功能提交到代码库中，我也会跟他们讨论打算先提交哪一部分功能。

如果想改变基础架构，要确保团队一直在做持续集成，而且应该选择他们在集成全部工作的迭代中进行改变。要小心：在团队将整个项目集成起来之前，在外人看起来，项目经理好像对项目失去了控制。（见11.2节“小乔爱问”。）

要是你不在乎浪费时间，迭代的持续时间就尽可以长。如果浪费整个项目的时间也没关系，那你就使用顺序式生命周期吧。要是不能接受浪费时间，那就将迭代时间限制在一周至四周。迭代越短，需求重新排序就越容易，而且更容易适应可能发生的变化；迭代越长，人们能完成的工作块也越大。

我不推荐使用长于四周的迭代，这很容易让团队成员患上“学生综合症”（见8.10节中的提示：帮助团队成员避免“学生综合症”），而且他们也更容易让不易估算或拆分的大块工作保持原样。长于四周的迭代会让团队逃避使用持续集成（见9.1节），大家就很难随项目进展看到已完成的工作了。

迭代要有最短持续时间。如果不能有在两天内完成一个迭代的规划，还是别做什么迭代了。这样就在规划上花去了太多时间，而没有充足的时间开展项目工作。要确保，相对于迭代的持续时间，规划用去的时间不会太长。

5.8 尽可能使用“小石子”进行估算

附录B中提到的“小石子”就是大任务被拆分成的小任务，每个小任务的完成不会超过两天，通常一天足矣（见［Rot99］和［McC96］）。如果项目经理熟悉极限编程，用户故事就是“小石子”。“小石子”只有“完成”和“未完成”两种状态，不存在所谓的完成百分比。“小石子”的集合就是团队在安排日程时定义的任务，通常要用去几天或者几周的时间完成。在实际开发中，用“小石子”可以将任务切分成放在小时间盒（用时一到两天）中的任务。

所有的项目都能用“小石子”方式来估算任务和监控进度。如果团队中有人在估算时容易忘记任务，“小石子”正好发挥作用。创建“小石子”任务可以强迫团队成员记住所有要完成的任务和步骤。类似“管理软件配置管理系统”和“返工”这样的任务是最容易被忘记的。

尤其是在顺序式生命周期中，很容易低估返工的工作量。也许开发人员在根据经验进行估算时，很容易忘掉还要做测试的工作。也许测试人员会忘掉搭建测试环境需要的时间。

忘掉这些东西会导致项目无法赶上发布日期，或者达不到当前版本要求的缺陷水平，或者无法完成当前版本要求的功能集合。使用“小石子”可以帮助团队避免这些问题。

“小石子”方式在敏捷生命周期中用得很多。（否则团队如何能在一周内完成有用的工作？）但是在其他项目中，特别是使用顺序式生命周期的项目，“小石子”不常用。原因之一是，在项目开始时就定义出所有的小任务毫无意义。在基于每天或者每周的规划中使用“小石子”方式是有用的，但对于整个项目的规划并无裨益。

如果知道自己要做什么，“小石子”方式很有用。对于项目的大多数任务来说，项目经理都知道要做什么。可如果不知道要做什么该怎么办？有什么应对之策吗？

5.8.1 当任务不清楚时创建并使用“小石子”

如果项目经理在管理新产品的研发（这个产品从未以任何形式出现过）或是研究项目，就得使用“小石子”的方式而且要适应它。

这种情况下，不要用时间盒，而是用问题来知道一项任务何时完成。不管是新产品开发还是研究项目，它们都有要回答的问题。团队成员或从事这些任务的人要提出特定的问题，还要知道如何找到答案。一旦团队成员回答了这些问题，接下来的难题就不是要做什么，而是该怎么做。团队就可以把要做的任务变成“小石子”了。

5.8.2 如何得到“小石子”

项目中每个人都可以划分自己的“小石子”——每个人都负有为项目做贡献的责任。项目经

理、技术带头人都不必定义这些小任务。如果项目经理、技术带头人或架构师试图这么做，其他的技术人员不会采纳他们的建议。因为为其他人产生“小石子”就相当于微管理。用微管理的方式去管理人并不合适。如果他们不知道如何产生“小石子”，可以指导他们，但是不要规定他们应该怎么做。

提示：避免使用“小石子”进行微管理

没人喜欢被管得过死，毕竟我们都是职业人士。很多人一开始很难相信，“小石子”可以使你摆脱微管理的困境。因为项目团队把他们的任务划分成小的增量，一个任务只有“完成”或“未完成”两种状态，所以就不必通过微管理反复检查任务的状态了。项目状态在任何时候都是显而易见的。（有些项目经理或资深管理层人士喜欢干涉项目以示帮助，要是你跟他们一起工作，那“小石子”就帮不了你了——什么都不行。）

5.8.3　为什么使用“小石子”

项目的生命周期越像是顺序式的，或者任务的时间越长，“小石子”就越能起作用。当项目经理让团队成员以“小石子”的方式安排日程时，每个人都能理解项目的任务及其之间的相关性。“小石子”可以暴露任务之间的依赖性。当团队成员使用“小石子”时，他们可以利用提前完成任务空出来的时间，其实这样的时间是满稀少的。而且“小石子”有助于创建更准确的日程，至少大家利用“小石子”进行开发的这段时间的日程会更准确。

所有这些好处都会减少日程安排的风险。

铭记在心

- 绝对不要提供确定的项目结束日期。
- 任务越小，估算起来就越容易。
- 寻求估算的准确性，而不是精确性。

第 6 章

识别和避免日程安排游戏

6

即使项目经理自己努力做好估算、规划和日程安排工作，你遇到的出资人、管理层和团队成员还是有可能视日程安排为儿戏。项目经理要把这些人带回现实，不过首先要学会识别这些日程安排游戏。

所有的出资人和管理层都会逼你在日程安排上做出一些让步。即使你制定的项目日程已经相当合理了，他们还是会玩这样的游戏。不过他们抗拒的方式很容易识别，很少脱离几种固定的模式。项目经理只要能够识别出他们所玩的游戏，就可以更容易地掌控项目，得到理所应当的产出。

6.1　给我一块石头

克里夫与团队一起，用一周时间制订出了项目日程。他们完成了“哈德逊湾式启动”（见4.2.4节），并且确定已经识别出了主要的技术风险。他将风险和日程安排告诉了他的上司诺姆。“你就不能再早点完成项目了吗？”诺姆的一句话将克里夫送回了团队，步履蹒跚。

克里夫与团队又花了一周时间修改时间表，得到另外一个日期。他走进诺姆的办公室，说道：“如果你能在这里和这里为我提供更多的人手”，他指着几个里程碑，“我就能用一个月时间完成项目。”诺姆皱着眉头说道：“还不够好。我需要这个项目早点儿结束。”克里夫叹了口气，又回到项目团队中去了。

又过了一周，克里夫拿着另一个日程来找诺姆，“好吧，这就是我们力所能及的结果了。”克里夫说。

诺姆几乎连看都没看，就说道：“但是还是不够好。”

克里夫暴怒道：“**你到底想要什么？**”

“给我一块石头”（见图6-1），这就是诺姆玩的游戏。不管你制订出什么样的日程，你的出资

人总是希望项目能更早完成。你只会发现：出资人不会认同你提出来的每一个截止日期——你的日期总是离他们的期望值很遥远［BWe01］。

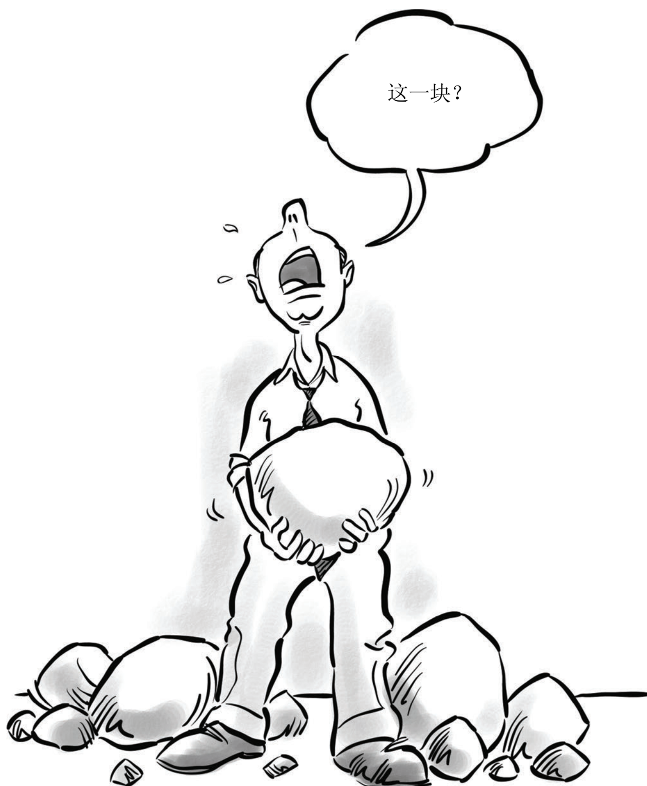


图6-1　给我一块石头

当“他们”希望项目能更快交付，但是不告诉你何时需要或为什么的时候，就会玩“给我一块石头”游戏。如果他们告诉你期望截止日期，你就能告诉他们能完成哪些工作。如果他们告诉你原因，你和团队也许就能想出一些创造性的解决方案来满足他们的要求。

讲求实效的项目经理自有应对之策，其中就有克里夫采取的谈判策略。一旦谈判失败，或者看起来永远难以成功，不妨试试下面的方式。

- 在试图取更多石头之前，先提几个问题：你喜欢短的日程，还是长的日程？是要更多的人，还是更少的人？要是少实现几个功能会如何？先知道什么是最重要的，这会帮项目经理产生更加合理的解决方案，多少也能为你的谈判做些准备。
- 找出是什么原因促成了他们期望的截止日期。探寻出这个项目的战略原因，并搞清楚“成功”的真正含义。

- 要让出资人明白你做出的选择以及背后的原因。也许出资人会有更容易操作、更快实现的好主意。
- 为你提供的日期说明信心范围。很可能管理层不明白你的估算意味着什么，而且你也有可能不理解他们所要的东西。
- 在提供日期时要说明发布条件，这样你就可以提一些问题，了解他们对于发布版本的质量和功能的要求。我们可以让这个功能的性能极其出色，但是就得先放下那个功能，这样做可以吗？用户们可以接受带有更多缺陷的产品吗？

在一个组织中，“给我一块石头”的游戏会反复发生。很多时候，每个项目都会发生这种状况。如果你总是碰到“给我一块石头”的问题，考虑采用下面的实践。

- 制订排好优先级的产品代办事项列表。见16.6节。
- 逐个实现功能。如果能够让出资人了解更多的项目进程，他们就不会那么纠缠截止日期了。
- 使用短小的时间盒（持续时间少于四周），这样出资人就可以看清项目进程。如果你每隔几周就能展示出项目的有价值的进展，截止日期就没那么重要了。你可以开始讨论何时实现哪个功能，他们对质量的要求又是什么。

6.2 “希望”是我们最重要的策略

几年前，一位资深经理打电话给我，说道：“我们有个项目出问题了。在启动的时候，我们充满了希望，但是现在看来已经不可能了。”我问了几个问题，发现他们之前从来没有做过类似的项目。相对以前，这个项目的规模更大，使用新的开发语言，基于新的平台，而且日程安排更短。

整个公司的未来都押在这个项目的成功上，问题是它比以前做过的项目都要复杂，而且要求也更高。他们唯一的策略就是“希望”（见图6-2）。

他们没有安排任何关于项目所在领域、使用的开发语言或新操作系统的培训。他们对这个项目希望达到的时间要求，也是以前从未做到的。

仅有希望，不足以交付一个成功的项目。①

讲求实效的项目经理会按如下方式做。

---

① 我是从以斯贴•德比（Esther Derby）那里听到这个游戏的。



图6-2　“希望”是我们最重要的策略

- 识别风险并记录下来。风险可能来自于技术（新的开发语言、新的平台）、日程安排（时间过短、人太少），很多时候两者皆有。
- 不到万不得已，不要选择瀑布式生命周期。为什么？因为你没有任何数据足以成功支持瀑布式需要的前期计划过程。要是你从未做过类似的项目，用迭代进行原型化，或者通过迭代开发几个功能，看看会是什么情况。
- 可以用“哈德逊湾式启动”（见4.2.4节）看看是不是能做出些什么东西。要使用即将使用的新的开发语言、操作系统、数据库以及类似新技术，这样做效果尤其好。“哈德逊湾式启动”能够让团队了解要做的东西，而且可以揭示一些目前尚未发现的风险。
- 确保大家具备相关的技术能力，还有解决问题必备的领域知识［Rot04b］。有必要的话可以进行培训。让大家学习项目中要用的开发语言，这些投入比起白白浪费时间所付出的成本要低。
- 考虑所有工作都采取迭代的方式进行，特别是项目规划和日程安排。
- 由于缺少经验和专业知识，可以寻求相关的帮助和信息。跟团队成员一起商量如何能让别人了解他们的工作进展。
- 制订里程碑条件（里程碑也可以是迭代的）。在管理层复审会议上审查这些条件。即使管理层或出资人不愿意做复审，项目经理也可以主导这些会议。如果不知道怎么样才能让

项目正常运转，可以按里程碑周期性审查项目进度。

不要指望仅凭希望就能得到好的结果。

作为项目经理，你的工作就是要计划、再计划，并努力工作，以得到最好的产出。以下这些实践可以帮你达到目的。

- 使用有时间盒限制的迭代，这样所有的人都可以看到项目的进度。
- 使用速度图表展示项目进度。要让大家都能很明白地看到进度（或匮乏的资源）。这样一来，特别是在需要帮助的时候，这些数据可以拿来使用。

6.3 拒绝女王

有些老板就是不愿意面对现实。你告诉他们：“我们无法达到你在时间上的要求。”他们就好像根本没听见，看着你，然后告诉你：“我相信只要你上心，就一定能按时搞定。”你坐在那里哑口无言，他们却高高兴兴地走了，好像项目在他们给出的日期前一定可以交付。出现这样的状况，你就是碰见“拒绝女王”[①]了（见图6-3）。



图6-3 拒绝女王

① 我第一次听说“拒绝女王”，是从本森·马古里斯（Benson Margulies）那里。

有不少原因会引发“拒绝”。我所见过最常见的原因，就像上面提到的经理，他希望鼓励项目团队。有时，人们拒绝是因为他们害怕项目无法在截止日期之前完成，他们不会管你说什么，这就是鸵鸟效应。有时，公司高层相信：如果设定模糊或是不可能的日期，项目团队就可以早于他们自己预想的时间完成项目。

可以用下面的方式应对“拒绝”。

- 找出你的经理表示拒绝的原因。尝试用一些与上下文无关的问题（见1.5节），以此来理解项目背后的原因。比如，你可以问：“对这个项目来说，要怎么样才算成功？”
- 写下项目的风险及其潜在影响。用高、中、低来讨论严重程度和暴露程度，不要用数字。不拿你制订的日程当回事的人，对这些风险数字也不会认真。
- 展示你所能做到的事情，并测量团队在项目中的实际开发速度（见11.2节）。没错，这里用迭代是非常方便的，而且速度图表可以告诉别人项目的真实现状。
- 保证参与项目工作的每个人都有相关领域的专业知识。

如果管理层认为“拒绝”是鼓励团队的方式，建议他采取其他的鼓励技巧。通常，这意味着让他去干点儿其他对团队有益的事情（比如通过谈判缩小需求的范围），或者将他的注意力吸引到别的项目上去。管理层可能认为拒绝现存问题或潜在问题可以鼓励团队，其实这反而会对团队形成障碍。将他们的注意力转移到其他项目上是一种很有用的技巧，可以让他们不再总盯着你的项目。

只要项目经理能够接受现实，“拒绝女王”就不一定能造成灾难。

有个团队试图说服项目经理接受项目的现实。失败几次后，他们放弃了，并且决定自己组织起来，忽略项目经理的存在。他们不再参加项目团队会议，并且将项目经理的话抛在脑后。他们开发了项目的部分功能，并且产生了一些数据（不过不是速度图表）。几个月之后，大家都看出来项目经理根本不了解目前的状况，此人也因此被炒了鱿鱼。可这时，项目团队也已经损失了很多时间，他们很难再控制什么时候交付哪些功能了。

现实总是会在某个时候跟“拒绝”面对面，这不可避免，而且这就是“拒绝女王”为什么只是规划游戏，而不会总是造成灾难的原因。当管理层看到了真正的现实之后，项目经理要确保自己有部分可以工作的产品、可以展示的数据，让管理层可以跟你讨论接下来能够做些什么。此时也是讨论“有哪些事情可以不做”（见5.3节）的好时机，这样你就可以完成当前的项目，并且为后续项目做更好的规划。

如果项目经理总是遇到“拒绝女王”，那么在进行管理的时候，可以把下面的方法集成进去，大家也就可以看到实际进展了。

- 使用有时间盒限制的迭代，这样所有的人都可以看到项目的进度。
- 制订排定优先级的产品待办事项。见16.6节。团队可以按照功能的业务价值，从高到低逐个实现。等到出资人如梦初醒，终于意识到项目无法按他给出的日期交付时，团队也已经开发完成了若干高优先级的功能，这样项目经理也会得到一些有价值的东西。

6.4 把灰扫到地毯下面

几年前，我接到一个项目组的求助电话。等我参与的时候，项目正处于发布周期的中期。我跟团队成员一起识别出他们可以交付什么、应该交付什么、还有哪些应该推迟。项目团队后来也按可交付物列表完成了交付。

发布之后，我建议团队召开回顾会议，看看有哪些工作是在下一次可以改进的。副总裁却认为没有人能从回顾中学到东西。

他忘了我为什么要来帮忙，光注意到了大家的成功。他说："不过大家这次干得很不错，满足了我们对这个版本的所有要求。"

他的话把所有的问题一扫而光——特别是优先级的变更——灰都被扫到地毯下面了[①]。请看图6-4。团队里可没人觉得自己做得有多好。副总裁的话不再可靠了。项目团队成员感到疲惫不堪。这个版本一开始要求的有些功能不做也是可以的，如果项目团队一开始就能知道这一点，开发人员就可以集中开发最需要的功能了。



图6-4 把灰扫到地毯下面

① 我第一次是从伊丽莎白·韩德瑞克（Elisabeth Hendrickson）那里听到"把灰扫到地毯下面"的说法。

下面这些主意可以避免这个游戏。

- 把某个版本需要的功能先按优先级进行排序，再实现。（排序意味着1、2、3、4、5、6，诸如此类。）参见8.3节。
- 逐个实现各功能。如果按照架构进行开发，就会形成很多部分完成的功能，没有哪个是完全开发完成的。而且架构会随着项目进展演化，很多管理层也认识不到这一点。参见9.3节。
- 制订发布条件，项目经理在项目开始时就可以讨论当前版本需要的东西。参见2.3节。
- 如果总是遇到“把灰扫到地毯下面”的问题，可以试试下面的方法。
- 使用产品待办事项列表，功能按业务价值进行排序。这样，你就总是在完成最有价值（也就是最重要）的工作。
- 使用有时间盒限制的迭代，并逐个实现各功能。项目经理和团队成员可以看到进展，并且总是最先提供最有价值的功能。

要想利用上述规避策略，就得在项目开始时与项目干系人对话。这些对话很难进行。不过其有益之处在于：如果没有交付任何东西，没人会假装项目是成功的。相反，团队可以把精力放在为了成功交付必须要做的事情上，而且只做这些。

6.5　幸福日期

我遇到一些组织，他们有个不成文的规定：绝不讨论项目日程。这种情况我见得太多了。管理层想要一个日期，项目团队就会说：“当然，没问题，就圣诞节吧！”但他们不会明说是哪个圣诞节。

最后，当某个圣诞节到来之时，已经过了不少日子了，人们开始讨论日程安排。而这时项目已经错过不少里程碑的交付日期，甚至有可能几个早先期望的项目结束日期都已经过去了。

我曾跟这样一个项目团队共事，他们在5年之内没有达成项目日程中任何一个里程碑或其他任何要求。他们总是在反复制定优化的项目日程，却没有信心范围。最后，在资深管理层变动之后，整个团队被叫去参加高层管理会议，解释日程为什么会如此糟糕。

有一个经理说：“你们看，我想知道什么时候才能有所成果。咱们找个出发点吧。”此时，项目团队就知道他们必须要做出改变了。

我必须承认，长久以来，我很难理解人们怎么会陷入到这种日程安排游戏中。不过，我确实见过善于口舌的管理层，他们或威逼，或利诱，或用权力来“说服”项目经理以及团队，让他们相信可以满足管理层对“幸福日期”（见图6-5）的要求。这种所谓的“说服力”加上逃避困难话

题的文化，足以让项目经理适宜进行“梦想时间”或“幸福日期”日程安排游戏。[1]

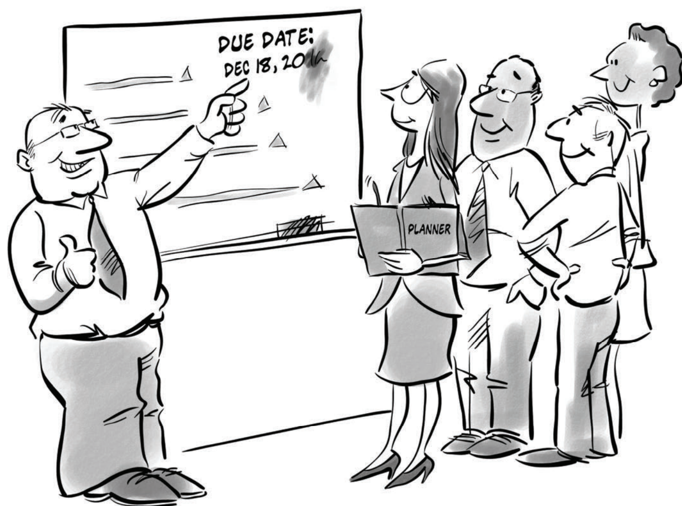


图6-5 幸福日期

“幸福日期”与“拒绝女王”有点儿联系，但是又不完全相同。有了“幸福日期”，组织中有些人（项目团队或项目经理）就可以让另外一些人（项目干系人）感到心安。从某种意义上来说，大家都不认为日程有讨论的必要。项目团队想抚慰项目干系人，项目干系人也愿意得到安抚。而在“拒绝女王”游戏中，项目干系人也希望得到安抚，可项目团队会坚持告诉他们现实。

要阻止这个游戏的发生，你必须与组织一起在项目层面上工作。对于项目，可以采取下列措施。

- 说明日程安排范围（见5.1.6节），特别是在没有使用迭代生命周期的情况下。
- 使用迭代式生命周期，并说明将会通过信心范围实现哪些功能，参见5.1.5节。（“到下个月为止，我们可以完成这十个功能，也许还能再多完成三个。我们会在本月底之前告诉你。”）
- 使用敏捷生命周期和经过排序的待办事项列表，参见16.6.1节。
- 即便采取按阶段交付的生命周期，也要使用短小的时间盒，这有助于人们不断取得进展，而且要让大家了解这些进展。
- 不要仅仅以里程碑日期作为项目的衡量标准。要想了解项目的真实状况，如果只使用单一的衡量标准（如11.1节所述）无异于饮鸩止渴。可使用速度图表，让每个人都能了解进度。

① 我是从蒂姆・李斯特（Tim Lister）那里第一次听到“幸福日期”这个词的。

不过，上述只是项目经理仅凭一己之力所能做到的。这个游戏揭示了组织的不一致性——大家只想彼此安慰，避免冲突［Wei94］。建设性的讨论（又名建设性的冲突）可以让组织更坚强。避免冲突和必要的讨论只会让组织变得更孱弱。

6.6　屁股着火

一天，一封来自公司大人物的紧急邮件把你招呼到公司。邮件里说："别弄那个项目了，赶紧开始做这个吧！"

可以确定，如果此事发生一次，它就还将多次重演。项目经理和团队都会周旋于几个项目之中，或是在两个项目之间不停切换。不管是哪种状况，项目经理都陷入多项目、多任务切换的泥潭中，而且整个团队亦是如此。项目经理知道自己无法取得任何成果，所有项目的紧急程度却在不断攀升、攀升、攀升……

当管理层害怕或无法一次将精力集中在一件事情之上时，就会发生屁股着火（见图6-6）的状况。有以下几种形成原因：技术人员过去就曾延迟交付，公司没有战略规划，或者公司的战略规划没有分解成足够详细的具体工作规划。①



图6-6　屁股着火

① 蒂姆・李斯特在与伊丽莎白・韩德瑞克谈话时，命名该游戏了。

项目经理可以采取下面这些行动。

- 按小时间盒迭代进行规划，在每个迭代边界都开始新的工作。要达到该目的，迭代必须足够短，比如一周或两周，这样才有机会开始新工作。
- 如果无法管理迭代，就按功能逐个实现，使用按阶段式的交付。
- 将采取这种策略的成本告诉管理层，让他们选择是要解决时间上的危机，还是要付出经过计算的额外成本。参见16.7.1节，了解如何帮助管理层计算成本和收益。
- 策略确定后，帮助公司检查相应的具体措施。可以这样问："根据该策略，我们会得到这样的结果。这真的是我们想要的吗？"
- 项目经理可以调整估算方法，让团队更容易达到最初的估算日期。如果团队无法满足估算日期，管理层可能会觉得别无选择，只能让他们去做别的事情了。保证大家都用"小石子"式的方式（见5.8节），而且尝试使用持续集成（见9.1节），这样团队就可能完成一些工作。
- 有时，如果客户觉得还不需要目前的产品，也会发生"屁股着火"的状况。管理层觉得团队没有完成这个产品的必要，他们想让所有的人或者绝大部分人去参与别的项目。既然如此，要让大家以短期迭代方式工作，而且在项目启动时就要知道是否存在会导致项目推迟的原因。还要确保管理层已经制订出了项目组合方案，而且目前也在管理该组合方案。请参见第16章。

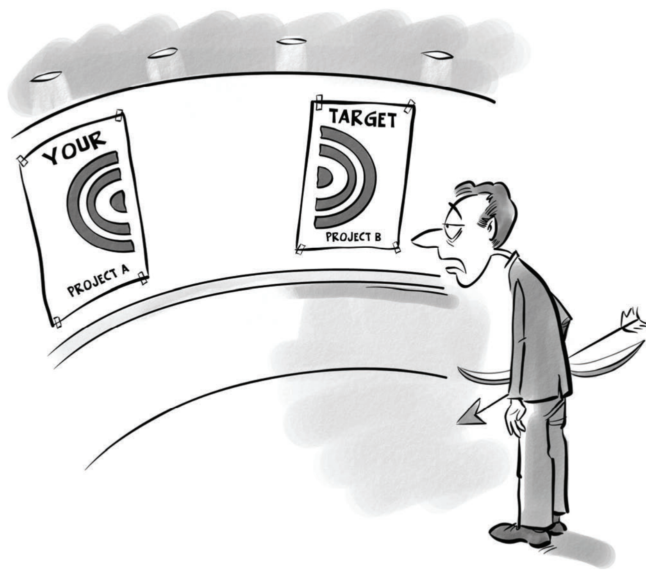


图6-7　分散注意力

"屁股着火"会浪费所有人的时间。不过，管理层有时就是无法改变他们的管理风格，或者

就是不相信同时进行多个项目会浪费时间。如果你陷入此类状况，不妨考虑如何创建单一项目的工作环境，这样你和团队都可以成功取得进展。

6.7 分散注意力

我曾遇到这样一个经理，他的原话如下：“我希望你能在项目A上花50%的时间，在项目B上花30%的时间，在项目C上花20%的时间。在空闲的时候，你能看看给老大的报告吗？”我答道：“什么空闲时间？”

“分散注意力”就是与多项目、多任务相关的游戏。如果管理层无法把精力投入到一个项目或是工程的策略之中，就会引发此游戏。他们不会对每个项目说“是”或“否”，也不会问“什么时候？”，而是对所有的项目点头称是。

项目经理可以采取下面这些行动。

- 与工程团队一起尝试16.7.1节中的解决方案，特别是有助于工程团队或管理层团队判定先做什么、后做什么的方法。
- 如果管理层坚持要你和你的团队同时进行多个项目，可为每个项目设定一周的迭代，并确保在每个迭代结束时有可发布的产品。我使用一周的迭代，人们就能在这一周内将注意力集中在一个项目上。一周结束时，到了周末，人们就可以改变关注点了，大家可以不再考虑眼前的项目，而是开始思考下一个项目。
- 如果无法管理迭代，那就按功能逐个实现吧，使用按阶段交付方式。要保证每个项目都有发布条件，这样就能完成每个项目最低限度的工作量。
- 就像在“屁股着火”中提到的一样，将这样做的成本告诉管理层，让他们知道要在时间上的危机和付出额外的成本之间进行权衡。参见16.7.1节，以了解如何帮助管理层计算成本和收益。
- 确保已经对需求进行过优先级的排定，而且可以赶紧完成一些工作。“分散注意力”游戏的发生，有时是因为一些项目干系人不相信团队能够按时交付。他们想让项目经理和团队同时做几个项目的工作，这样能让他们更快看到成果，并由此产生安全感。他们错了，却没有认识到这一点。

如果项目经理所在的组织经常“分散注意力”，那就准备采用一周一次的迭代吧。没错，这样做很困难。你必须跟团队成员一起，将需求打散成足够小的部分，他们就可以在一周内取得工作进展。如果不能帮助团队转向在短期内开发小批量任务的方式，你们就永远无法取得成果。正如15.3.2节的文本框中所提到的，团队会经常产生延迟。作为项目经理，你有责任帮助大家完成工作，同时帮助管理层不要陷入“分散注意力”的泥潭。

集中注意力意味着只在一点上集中

我曾为一家公司工作，在公司外开完了几天的策略会议之后，大家决定：我们要同时“把注意力放在五个方向上”。

这不是集中注意力。

集中注意力意味着将所有的精力都放在一件事情上。五个战略方向实在是太多了。

然而，对于公司来说，为了照顾到所有可能的客户，而把自己的精力分散开，去开发尽可能多的产品。此类状况太常见了。如果你所在的工作环境就是这种状况，赶紧采用短期迭代吧，越快越好。这样你就可以告诉管理层：“好吧，我们下周做那件事情，因为再有两天我们就可以完成这项工作了。”要想取得更好的效果，将迭代的开始和结束都放在周中，比如周三。这样的话，如果管理层在周末又突发奇想，他们也许还会给你几天时间完成现有的工作。

对于你保持注意力集中的努力，你的团队成员会感激不尽。

6.8　日程等于承诺

我们都知道，日程安排只是估计而已，不过是大概猜测罢了。项目日程是对于团队何时到达哪个里程碑、何时完成项目的最佳推测。日程安排并不是预言，仅是猜测而已。但是有些项目经理的出资人会将这个猜测视为承诺（见图6-8）。



图6-8　日程等于承诺

如果面对上述困境，可以问问如下两个问题：

- 你是否关心团队交付的东西？
- 你是否关心产品的质量如何？

要想让有关项目日程的谈话富有成效，就得讨论项目后面的驱动因素、约束和浮动因素；应该商量在截止日期之前，至少有哪些功能是必须要交付的，这些功能应该达到什么样的质量标准。如果相关人员还没有准备好讨论项目日程、功能集合和缺陷水平，那么任何关于日程应该作为承诺的讨论都显得为时过早。

当资深管理层希望得到承诺时，我喜欢用信心水平与他们沟通："我有90%的信心在8月1日发布。如果允许在10月1日交付，我有100%的信心。"我会告诉他们这两个信心日期之间我们必须要做的事情，也因此而深入理解了将日程作为承诺对我来说意味着什么。

我还喜欢使用"交付日期渐进法（date-for-a-date）"。"我可以告诉你，下半年发布没问题。现在我可以把时间限制在某个季度（给出一个日期），然后可以限制在某个月（另一个日期），接下来（给出另一个日期）就是我们要发布最终版本的日期。"

不过我最喜欢的技巧还是使用有时间盒限制的迭代（持续两到四周，不能更长了），同时使用按优先级排好顺序的待办事项列表。这样，就可以应对任何时间的发布要求。项目经理明白，每个迭代的成果都可以投入正常使用。项目经理也知道，最重要的需求已经先实现了。如果管理层想进行产品发布，没有问题，因为产品已经整装待发。开发人员不必再做出任何承诺，反而是项目经理需要从提供需求的人那里得到承诺，因为他们要说明哪个需求在何时需要。

如果有人要项目经理承诺一个日期，那就考虑如何组织项目吧。试试用迭代，或者尝试用"逐步逼近的日期"，要不就用带有日期估算的信心水平吧。但是不要只承诺一个日期，这简直就等于是为墨菲敞开大门了[①]。你可以承诺一个日期，但是总会发生一些意外，让你的承诺变为水月镜花。

6.9 到了之后，我们会知道身处何方

有一个副总声称自己有"多动症"（Attention Deficit Disorder）。他不会要求不同的日期，实际上，他在不停改变项目的目标。一开始，目标是一个特定的功能集合。而从开发速度图表中可以看到，团队无法按时交付。这位副总就改变了目标，要提升某些功能的性能。但是性能提升在技术实现上很有难度。几周之后，他又决定把重点放在提高可靠性上了。

① 作者此处是指墨菲定律（Murphy's Law）会发生作用。——译者注

让我觉得有意思的是：他在做项目经理时没有这个问题，一点儿都没有。那时他很明确每个项目都有一个确定的目标，而且他会尽量注意不让高层改变项目的目标。可当他自己处在高层之后，却不能让项目有明确的目标，这些项目也就很难完成了。

如果高层变更项目的目标，或是对于项目应交付的产品有更好的主意，或是对于项目脱离正常轨道有了更好的想法，就会发生这个日程规划游戏。我曾见过有些不靠谱的项目经理，他们就像高层所做的一样，导致项目脱轨，“哎，你看，那样做是不是看起来更好啊？咱们就那么做吧。”见图6-9。

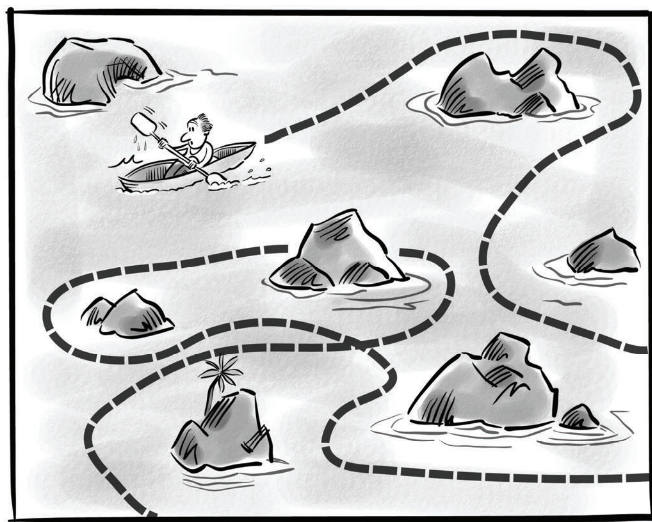


图6-9　到了之后，我们会知道身处何方

该游戏与“给我一块石头”不同。在那个游戏中，项目的目标不会随时间而改变，出资人想要的是更短的日程。在“到了之后，我们会知道身处何方”这个游戏中，出资人不想改变日期，而是希望改变项目目标。

当管理层无法或不想决定产品的方向时，这个游戏就发生了。

这就像是一个男孩总在等下一个最漂亮的女孩。他跟这个漂亮的女孩儿约会一段时间，如果另外一个美丽的姑娘出现了，他会抛下这一个直奔新的目标。

无论如何称呼这个游戏，其效果都是一样的。项目团队无法将全部精力投放到可以交付的一个产品上。项目总是在变换目标。最近一个我以咨询顾问的身份接触的类似项目中，一个高层管理人安慰我说：“等我们到了之后，就知道我们在哪里了。”

项目经理可以考虑如下建议。

- 确定已经有书面的项目愿景、项目目标和发布条件。就愿景、目标和发布条件得到大家共识。项目经理知道未来的方向、还要做哪些工作、做到何时就算成功。组织的其他人也要对这些了如指掌。
- 如果管理层不愿意定义愿景，那项目经理就要担起这个责任。完成后把它公布出来，并坚守这个目标。如果做不到，就赶紧结掉这个项目，再用新的目标启动新项目。
- 如果项目会持续很长时间，就用迭代来组织项目。每次迭代完成后评估项目进度。如果已经达成了足够多的目标，就结束这个项目，再启动新项目。要用短的（不超过四周）迭代。

有些情况下，上述措施都不能发挥效用，因为有些高层会干扰项目经理的工作，他们绕过项目经理，向团队分配其他任务。如果类似情况就在你身边出现，要跟高层沟通，告诉他们你能给他们带来哪些好处。如果他们不听，或者无法停止自己的行为，要记得自己并没有签卖身契。请参见7.7节。

项目需要清晰明确的目标，整个项目团队要把精力都放在这些目标上。不要允许别人让你的项目失去方向。否则你就会失去目标，到最后只能发现自己陷入一片迷茫！

6.10　日程安排工具总是对的，又称为梦幻时间日程

巴尼是一个项目经理，组织的高层只知道瀑布式生命周期。他们觉得迭代式的做法就是浪费时间。他们希望在项目的第一周就看到甘特图，这样项目经理就可以按照甘特图管理，一切都能按部就班进行。如此一来，无可避免的是，要是巴尼报告说项目没有按计划进行，有些高层就会这么说：“哎，按照日程安排，你的进度应该到这儿。没能按照计划进行，你是怎么回事啊？”

决策层对于项目的了解并不深入，他们不知道，人们在项目中是根据经验来思考和行动的。他们相信，关键路径会永远保持不变，任务安排顺序也一直大体相同。

发生如此状况，原因在于：一直以来，决策层看到的报表是由已经完成的工作、销售数字或其他数据构成的，这些数字反映的是在*过去*发生的事情。然而，项目日程是对于工作*未来*进展的猜测。该日程游戏也被叫做“梦幻时间日程”，参见图6-10。[1]

漂亮的甘特图会掩盖这样一个事实：日程安排只是猜测（见5.1.11节）。甘特图会让人们安心于日程安排，从而忽略对现实的检查。

如果项目经理面对这样的管理层，可以参考下面的建议。

---

① 艾丝特 • 德比和我在一次谈话中提出了这个命名。

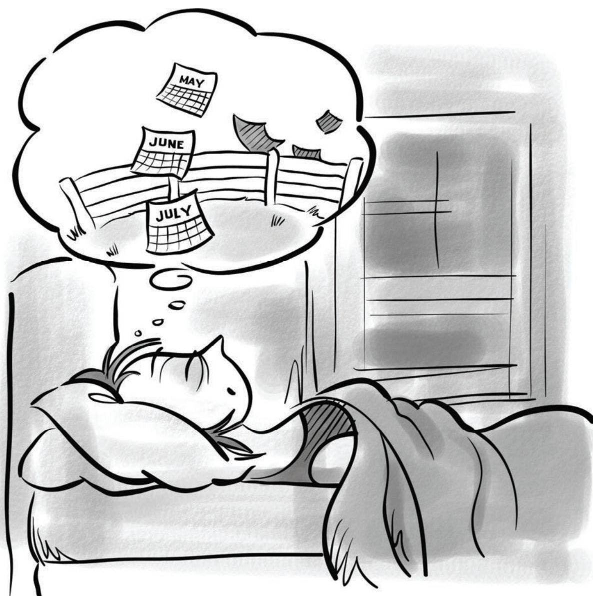


图6-10 梦幻时间日程

- 制订波浪式规划（见5.6节）。你只需规划出前几周的详细工作，还有主要的里程碑。过了头几周，要是你还不能提供详细的日程，人们就会觉得你无法预测未来。可以再制订一个新的日程，其中带有已经完成的任务、下一个波浪的工作、每个月更新后的里程碑。这样就可以告诉别人做完了哪些工作，而且不必束缚于一个无法实现的庞大计划之中。
- 使用低技术含量的日程安排技术，比如黄色即时贴式日程安排（见4.3.1节）或是墙上的卡片。还可以邀请决策层复查项目日程。
- 提供带有信心水平的估算（见5.1.5节），而不是用甘特图。
- 使用有时间盒限制的迭代，而且每次只规划一个迭代能做的工作。测量团队的开发速度。过了三个迭代之后，项目经理也许可以知道足够多有关速度的数据，这就可以预测项目剩余的日程了。

人们之所以坚信日程安排工具的正确性，是因为它假定估计出来的日程可以很准确。而问题在于很少有日程是准确的。很多可以很精确，比如“我们会在周三下午3：32发布产品”，但是却不准确。因为日程只是估算而已，让它看起来很漂亮，并不能改变这个现实，而且日程总会有误差。

发生这个游戏，日程安排工具并没有错，关键是使用工具的人对其过于相信。作为项目经理，你必须选定最有效的技巧安排日程，而且要跟别人说明这个日程安排。如果确实有作用的话，用日程安排工具也是可以的。但是不要只因为它能做出漂亮的图表，就对其坚信不疑。

6.11　我们必须拥有这个功能，否则就完蛋了

老板给项目经理曼尼打电话说道："曼尼，我们得聊聊你的项目了。"曼尼答道："当然，有什么问题？""嗯，如果这次不加入这个功能，我们就完蛋了。大客户们不会买这个版本的。"曼尼叹了口气，说道："我跟大家说一下吧，回头告诉你结果。"

曼尼转头跟项目团队解释了这个情况，大家同意在当前版本中加入这个额外的新功能。不过大家已经对按日程及时交付不抱希望了，顺便说一句，项目日程从来都没有改过。

在这个例子中，每个人都希望项目成功：老板、项目经理、项目团队。可他们没有权衡这样做给项目带来的影响，从而使得项目完全失去了按日程交付的希望。整个项目（和团队）就这么完蛋了。[①] 参见图6-11。



图6-11　我们必须拥有这个功能，否则就完蛋了

如果管理层和项目团队都是出于好意，项目经理可以尝试下面这些方法。

- 要求改变功能集合。你可以问这样的问题："有哪些功能是在这个版本里面不需要的？我看能不能想办法把你说的这个功能加进去。"
- 要求更多的时间。"如果你愿意推迟交付时间，我们可以加入这个功能。"
- 要求更多资金。"我知道你为什么需要这个功能，可与其他功能相比，要多花两周才能做

① 请参见：http://www.stickyminds.com/s.asp?F=S11829_COL_2。

完它。我们不能随便拿出来一个功能，再把这个功能放进去，还得再考虑整个版本。你想让我现在动手做吗？团队做估算、重新规划版本，要占用一些时间。咱们还是搞清楚这到底是不是你想要的吧。”

要想阻止“我们必须拥有这个功能”，有个办法。如果是按照功能逐个实现，而且按期提供可供发布的产品，项目经理就可以和管理层重新安排下个版本要实现功能的优先级。使用不超过4周的迭代，是阻止该游戏的最佳方法。用4周的时间盒，人们最久只需等待8周就可以得到新功能[①]。

要是每季度发布一次，比如使用“发布列车”（release train）方式，想得到新功能，最长的等待时间是6个月。这对很多管理层人士来说都太长了。而且，如果每隔6～9个月才发布一次，那别人就要等上一年甚至更久了。

如果你曾在组织内部遇见过“我们必须拥有这个功能”，使用带有时间盒限制的迭代吧，这样你就可以管控这些要求了。或者考虑使用“发布列车”（见14.3.2节）。参见16.6.1节，以了解如何使用待办事项列表来管理对更多功能的要求，以避免这个游戏。

我们假设你已经有了一个还算靠谱的项目日程。可是总会有出乎意料之外的事情发生，不是每个人都能完成之前承诺的工作。在前一节中，你已经见到了出资人和管理层会玩的游戏。而我见过有些团队成员会玩这个日程游戏。再次强调，项目经理的职责是要让团队成员认识现实。

6.12　我们不能说“不”

作为项目经理，管理层想让你再向当前版本中塞进来一个功能。他们在玩“我们必须拥有这个功能”的游戏。作为一个负责任的项目经理，你告诉了团队成员新的功能要求。你已经跟管理层说团队会评估请求，而且会告诉他们新的发布日期，或是加入新功能带来的成本。

可是当你开始跟团队成员讨论这个功能及其带来的影响时，大家都不愿意说“不”、“还不行”，或是“这样做会带来新的成本”。大家两眼一闭，就把新的工作接下来了，而不去讨论时间和资金上的成本，还有对当前工作的影响（见图6-12）。团队这么做，有时是处于内疚，有时是因为没有估计到额外的工作真正需要多少时间。[②]

如果项目经理管理的团队不愿意说“不”，你就得帮他们学会表达不同意见。不过只对高层

① 假如新功能在一个为期四周迭代的刚开始阶段提出，那么这些功能只有在下个迭代中才能被安排开发，因为一个迭代开始之后，其中要开发的功能是不能发生变化的。这样，要得到完成的新功能就需要两个迭代，也就是8周。——译者注

② 请参见：http://www.stickyminds.com/s.asp?F=S11829_COL_2。

（或市场部，或是其他希望加入新功能的人）说“不”还不够。项目经理可以对任何事情表达不同意见。如果人们试图处理额外的工作，为了帮助他们管理这些额外的工作，可以考虑下列方式。

- 询问团队成员，看他们能否针对添加额外的功能制订计划。可以使用黄色即时贴式日程安排和相对大小方式，看看如何能做得到。如果可以提供一个大家都满意的计划，项目经理就要尽量实现这个计划了。
- 项目团队成员有时会说：“我们会加入这个功能，并且加班完成。”如果他们想加班，要建议他们用时间盒限制加班时间，并衡量加班的结果。项目经理可以说：“好吧，我们把工作按为时一周的迭代分开。可以加班一周，再看看我们的工作效率怎么样，疲劳程度又如何。过了第一周之后，我们再以正常的方式工作，再衡量工作效率。然后我们可以对比这两个工作效率，看看有没有引发诸如额外的变更或是缺陷的新问题。如果觉得加班没有效果，我们还可以按正常时间工作，到时候再看看是不是还能有其他方式做得更好。”
- 项目经理有时可以加入额外的人力来做更多工作。（不一定总好使。）如果组织中有人具备领域专业知识，可以融入到团队之中，而且团队也希望有这些人加入，项目经理就可以把这些人加入到团队中。不要加入对产品或团队不了解的人——想想布鲁克斯（Brooks）[1]法则，参见7.5节。

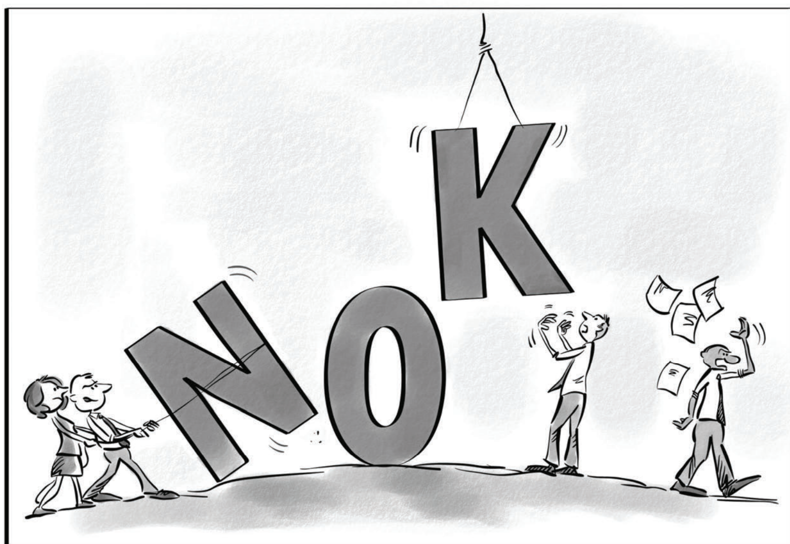


图6-12　我们不能说“不”

① 弗雷德里克·布鲁克斯（Frederick P.B Rooks,Jr）在《人月神话》中提出一个重要的法则：向进度落后的项目中增加人手，只会使进度更加落后。——译者注

如果上面这些方法都不起作用，项目经理就要帮助团队说“不”了。你可以用数据来抵消团队的内疚或是反驳完成公司要求的渴望。速度图表和迭代内容图表在这里特别有用。如果项目经理不能帮助团队学会说“不”，那大家就踏上死亡征途［You99］了。没有人希望这样。

6.13 日程小鸡

在逐个报告进度的项目进度会议上最容易发生“日程小鸡（Schedule Chicken）”的情况。[①]项目经理会问大家的进展情况如何。每个人都说自己在按日程安排进行，而实际情况却是没有一个人做到。每个人都在等别人眨眼睛并承认自己没能跟上进度。而且除非已经太晚了，没有人会承认自己错过了项目进度安排。参见图6-13。



图6-13 日程小鸡（树干上的文字是“日程”）

讲求实效的项目经理会有如下选择。

- 避免逐个进行的进度报告会议（见10.1.2节）。可以采取10.5节中的方法来判断项目真正的进展状态。
- 将任务分解成更小的部分，这样每个人每天都能交付一小块任务，最多两块任务（参见5.8节）。
- 按功能逐个实现。如果能有更多的人把精力集中在一小块可交付的功能上，就可以尽快完成这个功能，而且人们也更容易看到自己所取得的进展。在某些情况下，人们玩“日程小鸡”的游戏，是因为开始得晚（见8.10节中的提示）。定期看到产品的各个部分集成到一起，可以让大家都看到项目的进度，从而能够帮助他们知道何时没有取得进展。

① 我第一次讨论“日程小鸡”，是跟戴夫·史密斯（Dave Smith）和杰瑞·温伯格（Jerry Weinberg）。

- 考虑使用迭代吧，特别是使用敏捷生命周期。用了短期迭代，就不必再坐下来开每周的小组项目进度会议了（而且这个会议绝不应该开）。有了每日站立会议，人们就不能掩饰自己的真实进度了。

6.14　90%完成状态

许许多多的知识工作者，特别是技术人员，都未学过如何估算。就算是尝试过估算的人，他们也过于乐观了，总是会过低估计一项任务需要的工作量。要么得不到任何对估算的反馈，所以他们不会知道自己的估算是不准确的；要么他们预测不到一个任务中会牵涉多少子任务，比如准备测试环境或是签入代码这样的任务。在任何情况下，当有团队成员以为自己完成了90%的任务，而实际上还有90%工作尚未完成的时候，“90%完成状态”就发生了（见图6-14）。



图6-14　90%完成状态

我在职业生涯的早期，曾是“90%完成状态”的受害者。当时我在写一个数据库对话工具，要把一种数据库格式转换成另一种。我认为其中的数据是干净的，可事实恰恰相反。我以为我很清楚每个字段的格式，其实我不知道。我以为我对需求了解得很清楚，可随着工作的进行，每次处理数据库中的一条记录，我就会遇到更多的特殊情况，每一种都对需求有所变更。

最后我还是振作起来了。我开发了一系列测试用例，改变代码后我就会运行这些测试用例，慢慢地，整个开发工作开始取得进展。（要想知道更为现代的方式，可以阅读有关行为驱动开发

的资料。[①]）上司问为什么进度这么慢，我向他展示了当时的进度，并解释了我们在一开始并不了解的种种特殊情况。

作为项目经理，你应该指导大家，以消除“90%完成状态”。

- 帮助大家定义出自己的“小石子”。你可以跟对方坐在一起，然后提问：“要完成这项任务，你需要多久？这周要处理的细分任务都有哪些？
- 要让大家把自己的工作进度展示给你。这可能会揭示出他们代码中潜藏的问题（正如我的例子）、风险的列表、测试用例的列表，他们也可以在添加了一些代码之后，告诉你这些代码的临时意图。
- 教给大家如何跟踪自己的估算，让他们了解自己一开始的估算准确度是怎么样的。参见11.2.4节。

有些情况下，人们进入“90%完成状态”是因为他们的实现工作跨越了整个系统架构。如果项目经理让他们按功能逐个实现，同时以短期迭代方式开发，他们就会开始以更小的粒度对工作进行估算和实现。他们的估算会越来越准确，而且也能提高工作的完成度。

6.15　我们马上会变得更快

假定项目经理正在管理一个敏捷项目，或是按阶段交付的项目，或是其他生命周期类型的项目，总之这个项目可以用增量式的方式来构建系统。项目经理一直在测量团队的开发速度（或是实现的功能），可是进展速度没有达到预期效果。由于某些原因，团队成员对于如期交付很乐观（见图6-15）。

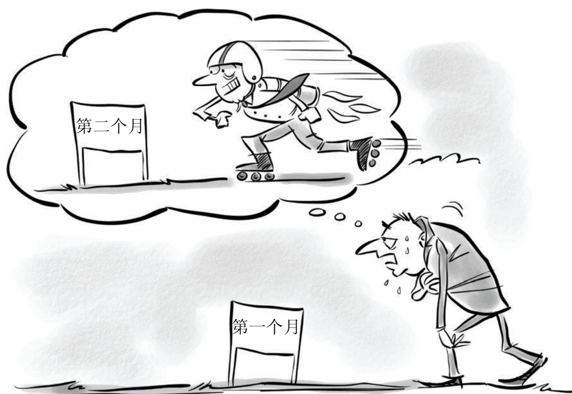


图6-15　我们马上会变得更快

① 请查看http://behaviour-driven.org/。

讲求实效的项目经理不愿意发出悲观论调或是冷嘲热讽，他希望让大家认识到现实，帮助团队成员看清真正的进度。说到底，也许他们还是可以赶上进度的。可以采取下面这些措施来管理信马由缰的乐观情绪。

- 与项目团队成员讨论进展速度。收集数据：是什么样的所见所闻，让他们觉得接下来可以比之前工作效率更高？
- 测量估算质量因子（见11.2.4节）。有些因素会让人们觉得自己一直在按进度计划开发，甚至超越了进度计划。要特别注意这些因素。
- 测量团队所做的每一件事情。确定每个人都全力投入到项目中，而且都为了按规定日期交付而开发必要的任务。如果有人在为其他项目工作，或是承担某个后续版本开发的任务，要马上中止这种状况。
- 如果项目中涉及硬件组件，要测量其挣值（参见11.2.3节），看看它是否能按进度计划进行。如果做不到（现有挣值低于应有挣值），重新规划整个项目吧。
- 在测量开发速度时，在团队完成第三个迭代之前，对于项目出现的任何整体速度数值都不要感到惊讶。因为只有到那个时候，团队的速度才有可能稳定下来。

6.16　令人恍惚的日程

下星期二，来自总部的大人物就要来视察了。或者你可能要在十个星期后做上市演示。不管怎么样，你要面对一个不可改变的日期、一系列充满野心或是不可能实现的功能集合，而且这些功能必须要在那个日期之前完成。无论是否已经测量过团队的开发速度，你知道团队是无法在截止日期来临之时完成所有的功能了。看起来，团队面对着日程已经处于恍惚的状态了。有时，这就是团队对于“幸福日期”的回应。参见图6-16。

首先，要创建项目的仪表板（dashboard，见第11章），再测量进度，从测量开发速度开始。然后考虑采取如下行动。

- 如果还没有用迭代式开发，赶紧把剩下的项目分解成迭代吧，迭代时间越短越好。如果在不可改变的截止日期前有十个星期的时间，项目最少要分解成5个迭代，最好是10个。每周一个迭代，这可以帮你不断调整工作的优先级。迭代的目标是要完成某个功能或一组功能，这包括功能的开发、文档、测试，以及其他产品对于“完成”的要求。如果上市演示需要在线帮助，除非一个功能具备了在线帮助，否则它就不算完成。
- 在一个迭代中要集中精力，每日站立会议对此会有所帮助。项目团队的目标必须放在要完成的工作上。当完成足够多的细分任务之后，就可以得到一个完整的功能或是产品了。不要让团队中的人们受其他项目、未来工作或是技术债务（见附录B）的干扰，除非技术

债务使得当前迭代的工作无法完成。

- 如果还没有准备好，就按功能逐个实现吧。这样可能会产生不完整的系统架构，不过不必担心。如果一个功能需要架构提供某些方面的支持，团队会实现的。如果功能上不需要，那就没有人会用它。



图6-16　令人恍惚的日程

通过使用“游击队式敏捷（guerilla-agile）”[①]，可以消除很多上述的日程安排游戏。如果能够以不超过四周的时间盒来组织项目，按功能逐个实现，随进度集成，并测量进展速度，项目经理就可以中止这些游戏。如果总是能以这样的方式管理，那就可以完全避免绝大多数的日程安排游戏。

铭记在心

- 日程安排游戏总是会发生的。项目经理的工作就是要识别它们，然后管理项目，让项目仍然可以取得成功。
- 绝大多数情况下，人们玩这些游戏都不是出于恶意。
- 即使没有恶意，日程安排游戏还是会拖项目后腿，使其停滞不前。

① guerilla agile是作者Johanna Rothman在Agile 2007大会上的演讲题目，指的是非正式的敏捷实施方式，演讲提纲下载地址：www.agile2007.org/downloads/handouts/Rothman_383.pdf。——译者注

第 7 章

创建出色的项目团队



你知道“完成”意味着什么，也在努力把人和任务组织起来完成项目。要达到这个目的，你需要团队中的人都能够胜任其职，还得让他们彼此紧密协作。总而言之，项目经理要创建一个出色的项目团队。

创建出色团队需要两个步骤，首先要招聘或引进所有需要的人才，其次要发挥他们的能力，一起工作，成为高效的团队。

7.1 招募需要的人

项目经理也许没有雇佣新人的权力[①]，不过项目经理一定要能识别团队需要的人才和技能。团队中应该有下面这些角色。

名称	项目中的角色
架构师	组织并指导整个系统的开发，包括测试系统
开发人员	设计、编写产品代码
测试人员	设计、编写测试，包括测试代码
技术文案	设计、编写产品文档
业务分析师	收集、编写需求
发布工程师	设计、编写、维护系统，包括与系统相关的任何脚本
项目经理	组织项目工作

项目可能还需要其他角色，比如UI设计师或是固件开发人员。项目经理应该知道项目中的技术风险何在（参见7.7节中的文本框），以确定哪些角色是必需的。

这些角色不一定都得由不同的人担任。举例来说，架构师也可以承担开发人员和测试系统架

① 如果能雇人，可以看看［Rot04b］。

构师的工作。而测试人员也可以是测试系统架构师。

不是所有的角色都像你想的那么清晰。可能项目经理只需负责组织开发相关的工作，而不管测试或是文档，因为与你一起工作的还有文档项目经理和测试项目经理。而且，也许团队的人都不受你的管辖（参见7.3.2节）。关键在于每个项目都要有能够胜任工作的人，不管他们是一个人担任多个角色还是怎样。

如果项目经理没有吸引、招募或淘汰团队成员的权力，那失败几乎无可避免。请参见7.7节。

提示：小心只会用PowerPoint的架构师

在我参与过的一些项目中，架构师就像一只海鸥。他突然出现，扔下一堆用PowerPoint展示的、不知所云的架构图，就像海鸥排下了粪便，然后就迅速离开了。他不会努力解决项目最难的技术环节：让产品可以使用他给出的架构，或是让架构不断演化，使得产品可以跟上发布的版本。[①]

不是每个项目都需要架构师。如果团队中没有架构师，要让出资人认识到，团队需要时间来评估架构，看看会出现什么样的模式。

有可能架构师会以咨询师的身份出现。不过也许会遇到很糟糕的情况：墨菲定律告诉我们，当我们特别需要这名架构咨询师时，他有可能在处理优先级更高的项目。

要是团队中的架构师特别喜欢画各种设计图，而不愿意写代码，也不能回答开发人员提出的如何把设计结果融入到现有结构中的问题，那此人也不算是一名真正的架构师［SH06a］。把这个人从项目中淘汰掉，项目后面的工作要把评估架构的时间考虑在内。要将缺少架构师作为项目的一个明确的风险，以便于管理。

如果项目团队拥有丰富的经验、不同的个性和角色，他们就可以更快地识别出风险，这对于项目经理更好地管理风险很有帮助。项目的风险越大，团队的多样性就应该越丰富。

7.2　形成团队凝聚力

项目经理不应该是唯一负责形成团队凝聚力的人，团队采取的某些行为或是某些行为的缺失，也有可能阻止团队形成一体。

就我所知，帮助团队形成一体的最佳方式就是让他们一起工作，而不是去参加什么团队拓展活动。如果大家有同一目标，彼此承诺完成互相依赖的任务，而且采取商定好的工作方式，这就

① 乔治·斯特帕奈克（George Stepanek）提醒过我，如果制造出工件的人不会留在项目中，不去看工件如何使用或如何转化成产品，此时就发生了海鸥现象（seagullishness）。

是一个团队。如果希望团队凝结在一起，那么就帮他们制订一些只有共同工作才能完成的短期目标。（13.1节中的方式可以发挥作用就是这个原因。）

可我以为我是在构建一个团队

——克里斯托弗，项目经理

我还记得被提升为项目经理的那一天。之前我参与了几个项目，但和大家还都不是很熟，即使我们一起参加了耗时大半年的项目。成为项目经理之后，我希望在团队建设上多花些时间和精力。

好几个周五之夜，我都会拉着我的团队成员出去开怀畅饮。不过不是每个人都能参加。有些人已经成家了，他们必须回去。

我试着用一天的时间带大家出去参加彩弹野战游戏。一个技术文案直截了当地拒绝了。我想她的原话是：“我讨厌枪，即使是游戏。”

我还认为举办飞镖比赛是个不错的主意。可是只有三个人感兴趣，而不是整个团队。

这可把我难住了。在一次项目会议中，我问大家该怎么做才能提升团队合作的士气。有个开发人员说：“就是这样。”我没明白，让他解释一下。

“在一起工作，解决问题——我觉得团队就应该是这样。我不关心其他事情，虽然很喜欢周五晚上的啤酒之夜。可是我想见到大家是如何一起工作的，而不是如何一起消遣。这样就能知道如何跟他们一起干活儿了。”

没问题，这我可以做到。在团队会议中，我特地留出了时间让大家共同解决问题。我还以不同的方式确保每个人都能与团队共同工作，让他们可以创建和交付彼此存在依赖关系的任务。现在团队成型了。

我再也不担心团队不喜欢那些活动了，不过他们确实不是很关心。哦，对于我想帮助大家形成团队凝聚力付出的努力，他们很欣赏，那些活动也不再是问题了。

7.2.1　好工具让团队有好的发挥

出现了问题，仅指望工具来解决是不可能的。不过，有了项目经理的指导，好的工具对于团队凝聚力的形成是无价之宝。要记住，工具本身并不能解决问题，但是好的软件配置管理（SCM）系统和缺陷跟踪系统（DTS）可以带来很多好处。

SCM系统对于形成团队凝聚力至关重要。没有它，就不可能知道源代码的状况。开发人员很可能会互相修改彼此编写的代码，或者就会有人认为：有些代码只能由某些人修改，其他人无权更动。这些现象会拖慢你的项目。有很好的、免费的SCM软件可供我们使用。（我就是用Subversion

完成本书的。）

如果使用敏捷生命周期管理项目，而且在任何时候都不会有太多未解决的缺陷，那么缺陷跟踪系统可以像索引卡那样简单。如果总是有5~10个没有解决的缺陷，那就需要DTS来辅助项目经理深入了解这些问题了，比如它们已经存在了多久以及目前的状态。

7.2.2 软件配置管理系统应满足的最低要求

现代的SCM系统可以做代码分支、打标签、自动合并多个作者的变化，而且允许开发人员在自己的个人工作空间（沙箱）中工作。其他SCM还有很多别的非常棒的特性。要是你的SCM不能满足这些最低要求，还是扔到一边去换个新的吧。

7.2.3 缺陷跟踪系统应满足的最低要求

好的缺陷跟踪系统可以为项目经理提供缺陷数据的多种视角：到现在为止当前项目和产品的缺陷、优先级和严重程度、缺陷状态、缺陷持续时间，还可以为缺陷报告添加其他数据。如果项目经理可以看到项目和产品的缺陷，就可以比较它们有哪些类似，哪些不同。你也许就能意识到，加入一些文档、指南或客户培训材料就可以消除大部分的缺陷。优先级是指开发人员或测试人员认为该缺陷应何时修复。严重程度意味着对于用户的影响有多大。缺陷的状态可以包括开放、关闭、重新打开，还可以有别的。如果可以对重新打开的缺陷排序，就能知道开发人员是否在进步（见11.2.9节）。有了缺陷持续时间，人们就可以知道应该分配多少时间做修复工作了。所有这些信息对于管控项目都很有用处。

如果你的DTS系统不能做这些事情，不妨先了解下这个DTS的模式。如果它的模式无法满足你要求，还是赶紧以旧换新吧。不过很可能不用这么做。

7.2.4 团队发展的5个阶段

当团队在一起完成某些有价值的工作时，其发展会经历5个阶段：组建期（forming）、激荡期（storming）、规范期（norming）、表现期（performing）和中止期（adjourning）[WJ77]。一个项目章程中规定好的、令人信服的共识，可以让团队进入规范期（见1.6节）。如果团队中有不少人都看不到这个项目存在的意义，他们永远都不会进入规范期，因为他们觉得项目与自己无关。

团队刚刚成形的阶段就是**组建期**。当团队开始共同工作并彼此试探的时候，就进入了**激荡期**。如果团队（或项目经理）可以让彼此就大家的行为意见一致，团队就进入了**规范期**，团队在这个

阶段可以把工作干得不错。当大家更加紧密地协作时，团队就能进入**表现期**。项目完成，这个团队也就完成了历史使命，进入**中止期**。

项目经理最多能参与两个团队

你是项目经理，参与到一个项目团队中，对项目负责。而且，你还是组织中其他项目团队的一员，他们的项目经理也在尽力完成自己的工作。对于这两个团队来说，你的立场可能有所不同，不过你总是团队的一份子。

要想让你和同事们可以协同工作，我所知道的最佳方式，就是将所在团队的项目经理（也许职能经理会作为项目经理出现）看做自己人，即使管理层并不鼓励这么做。

如果团队进入了表现期，再有新的项目，就可以把他们重新召集到一起。即使团队没有进入表现期，项目经理也许积累了足够的经验，足以带领下一个团队穿越振荡期，进入表现期。

项目经理可以促进团队经过这些阶段，但必须花时间留意项目的风险和问题，并与团队成员不断沟通。不要把所有时间都用来与项目团队无关的人开会，或是绘制甘特图。如果你需要详细的甘特图，找一个全职的项目日程安排人员来更新甘特图。

7.3　让组织配合你的工作

为了完成项目，最有效的组织会根据项目来打造组织机构。在这样的机构中，跨职能团队会向项目经理报告，项目经理管理每天的工作，管理团队与组织其他部门的互动。

不过很少有项目经理可以在基于项目的组织中工作。实际上，绝大多数项目经理都工作于职能组织或是矩阵组织。由图7-1可知：为什么在不基于项目的组织中，项目经理很难帮助团队形成凝聚力。每个团队成员都要对职能团队负责，同时还要对项目团队负责。这会减少项目经理的权力，让他无法推进工作，同时拖慢项目进度。

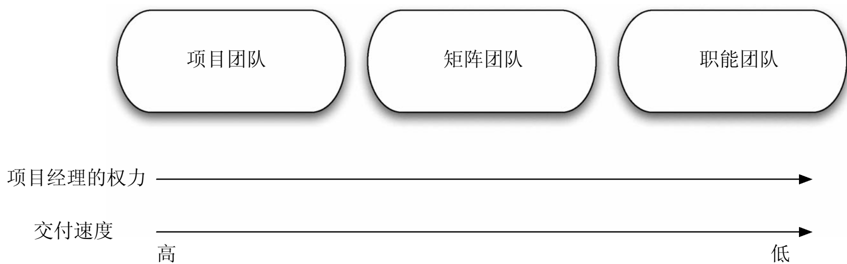


图7-1　不同类型项目团队对比

可以让职能经理参与到项目工作中，这样有助于团队形成一体。项目经理要跟大家一起工作，让他们明白自己被分配的项目任务，而不是职能经理分配的工作。项目经理要让职能经理知道：团队成员要先对项目负责，而不是对职能经理负责。这意味着项目经理要与职能经理讨论多任务的危险之处。参见16.7.2节。

7.3.1 以项目经理的方式管理职能团队

有些公司这样组织项目：开发人员组成一个项目团队，向开发经理报告，并视其为项目经理；测试人员组成另一个项目团队，向测试经理报告，并视其为项目经理；技术文案人员再组成一个项目团队，向文档经理报告，并视其为项目经理；以此类推。这种组织结构会形成彼此的间隔，并不得不使用阶段–关卡式的生命周期。

如果你就是一个类似的职能经理，也要管理所辖团队的项目工作，可以想一想是不是要继续这样的组织结构。项目会启动得很慢，大家会对它形成错误估算，而且总是赶不上估算日期，项目想成功结束也变得很难。项目中会产生非常多缺陷，因为项目在几个职能团队和项目经理之间传来递去。启动项目的项目经理（要么是需求经理或开发经理）会无法从头至尾控制项目，也许就不再负责完成收尾工作了。

你有好几个选择。不妨说服同事用管理工程的方式来管理项目，因为这样可以得到更好的结果。每个职能项目经理领取一些功能或是一系列需求，并使用跨职能团队来完成项目。这就像图7-1中表示的，整个项目团队从职能团队转向项目团队。

如果无法说服同事做出改变，那你可就陷入难以脱身（有人会说是不可能脱身）的泥潭了。要是组织不支持跨职能式的工作方式，可以尝试找一小组开发人员、测试人员和技术文案（他们都是项目团队的一部分）一起，按功能进行开发。

要是人们（包括经理和技术人员）都决定待在自己的小天地之中，看看是不是能找一个人作为总体项目经理，从项目一开始到最终发布，他都对项目负责。如果这些方式都不行，想想看你是不是还有必要留在这个地方吧。参见7.7节。

7.3.2 管理矩阵式项目团队

如果你管理的是矩阵式项目团队，团队中每个人都有自己的职能经理，而且你还是他们的项目经理。你的工作就是为他们分配任务，并确保任务可以完成。矩阵团队有个问题：成员的经理们有时会觉得可以让下属参加多个项目。也就是说这个成员可能不再是全职从事你的项目工作了。项目经理要具备谈判和说服技能。考虑使用项目组合（见16.1节），跟经理讨论多项目会带

来的延迟，让他们认识到多项目是不好的。

要跟职能经理说好，谁负责给团队成员反馈和指导。项目经理（或是技术带头人，或是项目中的其他人）要担起这个责任，对项目成员的表现进行反馈。

不过可能还有其他的讨论，比如职业发展，这是要由职能经理提供的。要跟职能经理分清楚，哪些人负责与团队成员讨论什么样的话题。

7.3.3　管理跨职能项目团队

如果项目中的每个人都能对项目忠诚无比，作为项目经理，你担负的责任就更大了。你要知道团队中每个人的工作方式，这样才能提供有价值的反馈，并对他们进行指导。

你也许不能直接管理每一个人。这种情况下，你的下属项目经理和技术带头人就必须知道如何给出有效的反馈和指导。

7.4　对必需的团队规模了如指掌

想在人手不足的情况下完成一个项目还不是最糟的。最糟的是，项目团队过于庞大，而管理架构又有所不足。虽然多于6人的团队总是会自动拆成几个小组，但是项目经理一个人最多还是可以管理9个人的团队。如果团队成员多于9人，根据可交付物，项目经理需要一些技术带头人或是其他项目经理的协助。

用9这个数字是有其原因的。考科伯恩［Coc01］表示，一旦到达这样的规模，团队的气氛就会发生变化。大规模团队成员之间的关系并不紧密［菲利普斯（Philips）和温伯格[1]所说］，即使他们的可交付物是相互依赖的也是如此，他们只是团队中的几个小组。特别是位于不同地点的团队，项目经理要花费更多精力来让团队觉得自己是一个整体［KS99］，而不是一个小组。

带领团队开发功能的人就是技术带头人。（如果必须按架构开发，他们就会带领团队开发架构中的部分功能。）如果要管理的是基于Web的应用，可能有一个基于功能的团队，他们可以用某个用户的账号登录，并访问用户的个人档案。特定领域的技术带头人要知道功能集合的性能、可靠性、数据库，还有可用性。

如果团队人数的确多于9人，他们可能已经创建了事实上的子小组。项目经理要给每个子小组指定一个技术带头人，这样才有可能管理规模过大的团队。

① 个体沟通。

责任与权威

没有哪个经理有足够的权威来做自己想做的事情。总是有职位更高的人。（即使你是CEO，也得向董事会报告。）即使名义上的权威很有用，也还是有所不足。

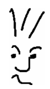
对于组织来说，如果项目的战略位置非常重要，不妨先斩后奏，去做项目需要的任何事情。从打听项目状态的人数以及这些人的级别，可以判断出项目的战略重要性高低。这些人的级别越高，人数越多，项目在公司中的战略位置就越重要。

要是项目从战略上讲不重要，就别在上面浪费时间了［RD05］。要是项目对于组织真的很重要，你就有权做任何需要做的事情。（对于要做的事情，你要有勇气、有信心。）如果项目不重要，你就不会得到足够的权力。去看看项目组合，找一个重要的项目吧。

即使项目很重要，项目经理也要使用自己的影响力来让人们完成你需要他们做的事情。要事先在组织中打好发挥影响力的基础。这样一来，等你需要帮助时，就可以号召其他人来帮你推进项目了［CB91］。我曾经让销售人员、服务人员、运营人员，还有市场人员来帮我推进项目。

要是项目经理已经在组织中工作了一段时间，就已经具备了影响力。如果你是个聪明又醒目的新人，人们也想让你成功（很多时候如此）。如果你以前没有经营过自己的人际关系（这就是组织中的政治），赶紧开始吧。“政治”不是一个肮脏的词汇，政治是你在组织中完成工作的方式，特别是在你没有足够的资源来独力完成的时候。

作为项目经理，你有责任树立权威，而不是等着别人给你权威。



小乔爱问……

我不能再加几个测试人员或技术文案了吗？

项目经理可以在任何时候加入更多的人。更有能力的测试人员可以帮你评估风险，或是帮助开发人员卸下一些寻找缺陷的责任。加入更多的技术文案，可以减轻开发人员编写产品文档的工作。

不过别欺骗自己了。项目启动后再加入更多人，项目会花费更多时间。

7.5　知道何时应该加人

项目经理想在什么时候向团队中加人都可以，可不一定能够达成预期的目标。

改变人员构成，团队会至少因此退回到振荡期阶段，甚至有可能到组建期。如果雇佣新人，团队的整体工作效率会下降几个月的时间，除非项目经理专门指定一个“伙伴”［Rot04b］来辅

导新人。（新人一开始不会有任何产出，如果所有的人都来帮新人，整个工作效率可能更低。）如果指定了“伙伴”，就只有这个“伙伴”的工作效率会受影响，而且新人也可以尽快学会如何开始工作。

如果项目刚启动，项目经理应该尽快加入需要的人手。每次人员变动都会对团队的工作效率产生影响，所以应该尽量一次性将人员配备整齐，避免成员变更降低工作效率。

如果项目进入到中期，加人就得小心。要记住布鲁克斯的法则［Bro95］：向进度落后的项目中增加人手，只会使进度更加落后。

项目快结束时，要避免新增人手。只有下面这种例外情况才能加人：项目的延迟已不可避免，而且当前的人手也无法完成项目。那就增加需要的人吧，然后重新规划。

7.6　成为出色的项目经理

可能你听说过这样的说法：一个音乐演奏家该如何才能到卡耐基音乐厅去演奏呢？只有练习、练习、再练习。要想成为一名出色的项目经理，也是如此。

不过，要想成为出色的项目经理是有要求的：为了与团队一起工作，你要提升人际交往技能；为了了解和管理项目的风险，要提升或维护足够的技术能力。

我将技能分为如下几类：人际交往技能（非技术素质、偏好和技能）、功能性（技术）技能、领域专业知识、工具和技术上的专业知识［Rot04b］。参见［WJC00］以获得不同的能力分类。

7.6.1　提升人际交往技能

项目经理的人际交往技能，也就是你如何与人打交道，对于项目的成败至关重要。基本上，所有的项目经理都需要掌握下面这些人际交往技能。不过，阅读类似的列表，收获因人而异。

- 倾听技能。项目经理要倾听团队成员的言谈，还要从他们那里了解项目的状态。
- 谈判技能。项目经理要争取资源、交换资源和信息。
- 写作技能。项目经理要能够编写计划，让每个人都可以理解计划以及做出的妥协。只有一些简单的列表是不够的。
- 以目标为导向。项目经理要能够完成一个项目，还得帮助大家把注意力都放在项目的目标上。
- 了解和尊重项目相关的工作人员。项目经理不必成为每个人的朋友，但是必须要能看出来别人什么时候陷入困境，什么时候出现了问题，什么时候一切正常运转。

- 能够应对信息不足的状况，也就是可以适应信息不足的情形，并且作出决策。我管理过的每一个项目，不仅仅是软件项目，信息都不是完备的，也就是都有某种程度上的不明确性。
- 能够管理细节。即使项目经理不是一个细心的人，也必须要学会管理项目的细节。
- 解决问题的技巧。项目经理要识别出哪些问题当前必须解决，哪些问题可以推迟，以及如何解决问题。“三种备选方案原则”（The Rule of Three）[Wei85] 是很好的问题解决工具。（“三种备选方案原则”是说：一种备选方案是陷阱，两种备选方案是两难的困境，提供三种备选方案就可以让相关人员开始考虑真地该怎么做了。）
- 辨别、寻找进度的障碍，并消除它们。

掌控项目的能力。要观察目前的状况，指出项目当前方向与你最初的规划有何不同，并可以引导项目进入新的状态。

7.6.2 提升功能性技能

在管理项目时，项目经理要能够使用不同的技术；在开始管理之前，项目经理还要能够学习不同的技术。

- 对于项目中的问题以及如何解决这些问题，项目经理不需要知道二者的具体技术细节，但是如果一点都不了解问题和解决方案的专业知识，项目经理就很难作出好的决策。参见7.7节中的文本框。
- 不懂技术的项目经理，不要试图遮掩自己技术上的不足。要敢于承认自己的不足，雇佣聪明的人，而且可以在了解项目的技术点时咨询这些聪明人。如果项目经理能够坦诚表明自己的弱点，并表示出学习的意愿，团队是愿意帮你成功的。
- 理解不同的生命周期模型，知道哪一种最适合你的项目。
- 能够安排项目的日程规划。
- 能够估算任务，或者指导其他人完成任务估算。
- 知道如何评估风险、管理风险。
- 清楚如何度量项目状态，以及如何报告项目状态。
- 知道如何处理已完成和未完成的工作，可以使用速度图表或是挣值（见11.2.3节）。如果没有上面这两种度量方式，要能理解软件配置管理系统中的数据和代码的状态。

7.6.3 提升领域专业知识技能

软件项目的项目经理要能理解人们收集和排序需求的方式，要知道如何了解设计是否完成、如何评估技术风险和日程风险，要知道软件配置管理系统的作用以及如何有效使用，还得知道测试人员能够提供什么样的信息。项目经理要能够从不同的审核活动中，帮助项目团队选出最适合

当前项目的审核活动。

这并不是说项目经理必须要知道如何完成这些任务，但项目经理应该知道如何组织项目各项活动，促成上述任务的完成。要想知道如何进行这些活动，项目经理要具备一些问题空间域和问题的解决方案空间领域的专业知识。

项目经理应该了解多少技术

项目经理不必成为项目的技术专家，但是应该了解开发用到的技术是如何解决客户问题的，还要知道团队用到的流程，以了解项目风险。

项目经理要了解足够的技术，这样他们在决策时就能进行权衡（或是帮助产品负责人做出权衡），判断发布的产品要包含哪些功能。项目经理越了解开发中的产品，就能作出更好的决策，或是指导团队作出更好的决策。

要避免下面两种极端情况：一无所知的项目经理；想成为架构师的项目经理。我曾遇到这样一些组织，他们认为其他行业的项目经理，比如规划各种市场活动的经理，也可以成为出色的软件项目经理。不行，绝不可能。项目经理要理解项目的流程，还要知道足够多关于产品和工具的知识，这样项目经理就可以评估风险，并在项目中加以管理。

以我的经验来看，项目经理作为架构师也不好。这样的项目经理了解流程和技术，但是会置项目经理的工作于脑后。如果项目经理专注于开发而不是如何管理项目，这就跟项目经理对于项目一无所知的情况差不多，项目也会经受打击，不过方式可能有所不同。

问题空间域的专业知识是指理解项目要解决的难题。解决方案空间领域的专业知识是指理解系统如何利用解决方案来解决项目的问题［Rot04b］。

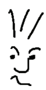
项目经理要能够快速获得对领域的理解，特别是理解需求（以及需求排序）相关的问题空间，还要知道解决方案空间有关架构的部分。如果不知道项目要解决什么样的问题，又怎么能知道项目何时算是完成了呢？再说，如果项目经理不了解架构，就不能了解技术风险。你可以不了解所有的技术风险，但是如果不了解架构，项目经理甚至都不知道问什么样的问题。

注意，这里并没有说项目经理要亲自阅读或是编写代码（或是测试）。成为好的开发人员或是测试人员，虽然有助于你了解软件项目的状况，却并不意味着你一定是好的项目经理。功能性技能是不一样的。当然，项目经理懂得的技术可以更多，可如果要是知道得更少，那项目经理就很难有效管理项目了。

7.6.4　提升工具和技术的专业技能

如果使用黄色即时贴，项目经理可能就不需要项目日程安排工具。不过我知道的大部分项目

经理都会使用这样的软件。项目经理得知道如何让工具符合自己的要求，或者有个助手知道也行。



小乔爱问……

项目经理需要PMP认证吗？

不需要。认证只能证明你很善于考试并且有些项目经验，但是并不能说明你在管理这些项目时有多么成功。

为什么我对PMP这么看不顺眼？因为PMBOK中的项目管理只能在顺序式生命周期中起作用，而且项目经理要有话语权。（PMBOK没有说必须用顺序式生命周期，但是很多仅接受过PMI培训的项目经理只知道这个。）很多项目，特别是绝大多数软件项目，使用顺序式生命周期模型毫无意义。使用命令与控制的方式毫无意义，因为协作方式可以更有价值，耗费时间更少，还可以得到更好的结果。

这不是说学习PMP认证就毫无价值。价值在于学习，并能将其中的知识应用到项目中。不过一张PMP证书并不能保证你就有项目经理的能力。这还远远不够。

7.7 知道何时该全身而退

一名出色的项目经理知道什么样的组织、团队和产品不适合自己。如果你发现自己身处下面的状况，问题可能不是出在你的身上。所在的组织可能有问题，而现状可能已经无法控制了。

至少，你总是可以有下面几个选择：接受目前的状况并心中有数，转换到组织中的其他项目，改变所处的组织。你有可能促成组织的改变［Wei97］。下面是应该注意的警告信号。

7.7.1 什么样的组织不适合你

你不能选择团队成员。很多项目经理的团队成员会随着他们转战各个项目。偶尔，有一个成员不能完成你分配的工作，或者不能融入团队。你也无权让他离开团队。

如果项目经理不能获得自己需要的人，先想想自己是不是解释过为什么需要这些人，而且要确保这么做。如果能够把成本或是收益与这个职位相关联，那就再好不过。我曾经要求给项目配一个发布经理，结果被告知：“JR，我们是不会把钱白白花在一名发布经理上的。开发人员非常清楚该如何签入他们自己的代码。”

我不担心代码签入的问题，我担心的是不同分支和不同平台的代码如何管理。我让每个人记录日志，记下自己使用SCM遇到问题时耗费的时间，还有必须自己手工写SCM脚本用去的时间，

等等。这样记录了一周，发现团队在SCM上用去了30小时。我把这个结果告诉我的经理，可他不相信。我就让团队再记录一次相关的时间。下一周，他们用了38小时。这下我的经理就同意了，我也雇佣了一个人作为承包商，后来此人成为了永久雇员。项目经理可能要用“人时”（person-hour）来说明项目的损失。

不过问题不仅仅是无法雇佣新人。我见过一些团队，项目经理备受一个团队成员的困扰，这位成员要么是无法完成工作，要么是无法跟团队融合。如果你处于类似的状况，要确认你已经首先提供了有效的反馈［RD05］。要是你已经事先说明了你要的结果，并且已经就工作或是个人的行为提供了反馈，那么让这个人离开团队就是有意义的。



小乔爱问……

项目经理能同时管几个项目？

我总是惊讶于那些看起来似乎几天都没合眼的经理。“我在管理三个项目，而且推进得都不够好。可是老板觉得我至少可以管三个项目。我到底应该能管几个项目呢？”

我希望能有正确答案。可最好的答案是：“视情况而定。”如果你的团队能力超凡脱俗，知道如何协同工作，可以在内部解决问题，能够自己消除障碍，而且在没有项目经理的情况下也可以遵守规范、监督工作、掌控项目，那你就可以管理这个项目团队了，因为他们不需要你，而且你还可以再多管一个普通的项目团队。

如果要同时处理多个任务，而且试图管理多个项目，你就得知道自己至少在一个项目上打马虎眼，如果不是所有的项目都打算蒙混过关的话。成功的项目经理可以一次管理一个项目。要想多管几个，你会把自己搞疯掉，而且基本上不可能成功。

我曾试图将工作状况不佳的团队成员与团队隔离开，可从未成功过。让这些人在身边转来转去，会令其他的团队成员焦躁不安，这就像是有问题的人反而得到了奖励，因为他没有从事相关工作，或是没有融入团队。

项目经理要让不合适的人从团队中离开。如果不能，想想你自己吧。也许是该换到别的项目，更有可能的是，离开这个组织的时候了。

出席会议变成行政任务。如果在开会时，高层总是盯着谁列席了会议、谁没有来，你也因此不能选择参加哪些会议，那干嘛还在这样的组织里浪费时间？要先打听下是不是有人在盯着会议的人员参加情况。也许你做出了与事实不符的假设。如果你的假设正确，还是收拾收拾东西走人吧。现在的这个组织很快就会被淘汰掉。

出资人让你不得好死。你告诉出资人，让他对不太乐观的项目前景有个心理准备，结果他跟你说：“那我就把你的头拧下来。”如果人们受到威胁，或是觉得项目的失败全都归罪于己，他们

就很难在工作上表现出色，起码我见过大部分人如此。

有些项目经理觉得这样的环境很能鼓舞人，我不这么想。要是你这么想，那恭喜你，希望你能更加强壮。你会需要它的。不过要小心，这是一种命令与控制式的环境，而不是协作式的。协作是绝大部分项目成功的必要条件。

出资人坚持认为人们都应该同时处理多个任务，而且不愿意做出改变。你已经解释了多任务会带来的延迟效应，而出资人仍然无法想清楚哪个项目的优先级最高。而且，能否成功交付项目对你来说至关重要。

如果拿着高薪的这帮人不愿意作出关于项目组合的决策，那你为什么还愿意管理一个注定失败的项目，并将自己置于毫无获胜希望的局面之中呢？你不必如此，还有别的选择。

别人想让你承担一些开发相关的工作。项目团队除了你之外，还有三个人，而且别人觉得你应该写代码、测试、写文档以及其他的工作。反正人家觉得你处理完项目管理的工作之后，应该做一些技术和开发上的事情。

咳咳，如果你担任项目经理的团队成员多于三个人，而且你也兼任他们的经理，那你是不会有闲暇时间的［RD05］。（如果只有两个人，你还有可能干点儿技术活儿，只要你能确保自己的工作永远不会处于关键路径上。）要是团队成员以矩阵式方式加入项目，而且他们只从事与项目相关的工作，并可以按时完成，假定有5个成员，那么你也许还能做一些没有处于关键路径上的技术工作；要是少于5个人，那么你所负责的任何技术任务都将位于项目关键路径之上。（如果所在的项目应用敏捷式生命周期，你作为教练或是Scrum master，而且团队已经做到了自行组织和管理，那你每周就能腾出来更多时间干技术活儿了。）

我看到的情况是，项目经理没有管理项目。他们在编写代码，但是没有管理风险，他们没有留意测试的进度，他们没有产生仪表板上的数据，他们也没有评估版本发布条件。如果你在一个喜欢开会的公司工作，还得把所有的会议加到工作列表上去。

如果你在管理项目的话，要帮助大家看清目标、移除障碍，还得监控他们的进度，你是没有时间从事技术工作的［DeM01］。

管理层强制切分项目工作。假如你是开发经理。你启动了一个项目，努力工作，并声称开发已经完成，并向下一个项目进发。你的同事——测试经理——开始测试工作。两天以后，他冲进你的办公室，跟你说："你的开发人员活儿没有干完。项目是不可能在四周之内完成了。你得把你的人都给我用，好修复缺陷。"可是你已经把人员都分配到另一个大项目中了。现在怎么办？

你们都是职能兼项目经理。两个人都不能完全对项目负责，而且你们有人（很可能是测试经

理）会因为延迟发布而遭受责骂。这并不是测试人员的错，也不是开发人员的错。当高层将项目组织成互相隔离的部分，而且作为职能经理的你对此毫无怨言，这都是管理层的问题。

在7.3.1节中，我提出了一些建议，读者可以尝试一下。不过对你来说，真正的问题在于：当组织不以结果作为导向时，你能在自己的职位上发挥作用吗？要想一想：是不是值得花时间和精力来改变组织？还是干脆改换组织？[①]

所有的项目在启动时都面临资源不足。你是一个老好人。上司跟你说："南希，我们想让你启动这个项目。不过目前可能无法给你现在需要的人手、电脑、办公空间，还有笔。实际上，项目进展到一半之前，我们都无法提供你需要的资源。不过我们对你有信心。你愿意为了团队来接下这个项目吗？"

噢，成为拯救公司的英雄，这种感觉真得很好。不过当你卷起袖子，接受并不完整的资源并准备启动项目时，这么做对于团队其实毫无益处。在资源不足的情况下启动项目，是完全不可接受的。

如果项目经理不能帮助管理层发现项目的驱动因素（见1.2节），这个组织就不适合你。这个组织会继续利用你善良的天性，并将你推入必死的项目之中。别让自己掉进去。要么改变项目，根据驱动因素、约束和浮动因素来重新组织，要么走人。在资源不足的情况下启动项目，必败无疑。

你总是听见人们说"你没法融入团队"。项目经理的工作要求你将项目视为组织中最重要的工作，而且必须依此行事。如果你为了项目成功而推行的做法得不到认同，而且总是听到"你没法融入团队"这样的说法，那么也许这个组织根本不需要你。

工程经理的视角并不仅限于一个项目。不过，即使是工程经理也不允许组织对他们横加干涉。你是项目经理，你说了算。你对项目的成功负有责任，请参见7.4节中的文本框"责任与权威"。要满怀热情地实现这个责任对你的要求。

7.7.2　什么样的团队不适合你

对于管理项目需要的信息了解不足。项目经理可以没有管理过此类项目的经验，但是一定要知道团队如何工作，以及项目试图解决的问题。很难知道自己了解的信息是否足以管理某个项目，但是下面这些问题有助于你搞清楚这一点。

- 你知道团队是如何工作的吗？如果不了解他们的工作流程，也不知道配合团队工作的流

① Martin Fowler这么说。

程是什么样子的，你可能无法管理这个项目在流程上的风险。
- 你明白这个项目试图解决什么样的问题吗？如果不理解问题，你就无法帮大家创建有用的发布条件。更重要的是，你无法随着项目进度评估发布条件。

你的管理层非要给团队施加帮助，可你无法拒绝。我们都遇到过想施加帮助的经理。有时你允许经理“帮助”团队，而且无法拒绝。

如果你不能让这些管理层远离团队，就等于是无法帮助团队移除障碍。实际上，你是在为团队增加障碍，他们想完成工作就变得更难了。这不是在帮助团队，而是在伤害他们。

要敢于做团队的挡箭牌，让管理层的“帮助”落在你的身上，而不是团队。你不一定要接受这些帮助，只要聆听就可以了，不过这也挺糟的。

对于管理项目需要的信息了解过多。你的技术经验非常丰富，而且也曾管理过项目。不过这个项目用到的技术你非常熟悉。你能做到不去管技术，而全心全意管理项目吗？

如果你不能放下架构和设计等技术工作，因而无法集中精力管理项目，那就得做出选择了。要么管理这个项目，要么去做个技术人员。但是不要试图两样全占。干扰其他资深开发人员的技术工作，会让他们感到沮丧。你也无法移除障碍、管理风险，更不用说帮团队完成项目了。

7.7.3 什么样的产品不适合你

你不具备解决方案空间域的专业知识——其他人也没有。没有解决方案空间域的专业知识，也能把项目管理得很出色。可如果团队中没人具备足够的技术深度，那就没人能够了解技术上的风险。想想看是不是还留在这个项目中吧。

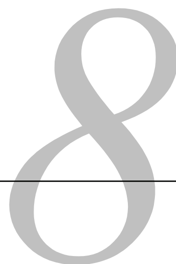
如果必须要跟项目在一起，可以采用敏捷式生命周期，再加上短期迭代。你需要做原型化的工作，还得按功能逐个实现，这样就可以知道团队是否交付了真正有用的东西。

铭记在心

- 项目风险越大，团队的多样化程度就应该越高。
- 提升多方面的技能：人际交往技能、功能性技能、领域技能，还有非技术技能。
- 知道何时应该离开。

第 8 章

掌 控 项 目



项目经理已经编写了项目章程，也有了相应的规划。团队知道“完成”的含义，也有了项目日程。大家的工作正常进行着。项目现在正式进入了中期，作为项目经理，你的职责就是掌控项目，使其到达成功的彼岸。

如果还没有制订出项目的仪表板（见第11章），现在赶紧动手。定性和定量的度量方式都需要，这样才能反映项目的真实状态。一旦真实状态清楚了，项目经理就可以决定如何把握项目的方向了。

掌控项目意味着要寻找风险和管理风险。通过组织项目来发现它的节奏，这是一种发现风险的方式。任何扰乱节奏的事情都是已经出现的风险，任何可能扰乱节奏的事情就是潜在的风险。通过积极地管理风险，项目经理可以保证项目不会偏离正确的轨道。

8.1 掌控项目的节奏

节奏，每个项目生而有之。有些项目节奏混乱，进展缓慢。有些项目就像坐上了火箭，倏忽间，团队就完成了更多工作。所有的项目都有节奏，不过它会随时间而变。观察节奏是项目经理的职责，你还要能看出来，那些实践是否能够帮助项目建立并维持合理的节奏，保证项目取得成功。

项目的生命周期越接近顺序式生命周期，项目的节奏就越有可能随阶段发生变化［Rot04a］。在早期阶段，项目看起来就像遭受了很多痛苦，项目经理也不知道需求何时冻结、设计何时稳定。有些项目会“神奇地”找到出口，在进入实施阶段后，节奏逐渐显现出来，而且日趋稳定，因为该做的决策已经做了，而且得以实行。假定没有人手方面的问题，项目就可以顺利完成。如果项目总是混乱不堪，项目经理就会在整个生命周期中都找不到方向，也难以发现令人舒服的节奏。

在敏捷生命周期中，迭代规划阶段也许会察觉到混乱的状况。不过迭代规划只会持续几个小

时，团队在迭代开始后可以找到像鼓点一样的节奏。（如果你曾见过敏捷生命周期找不到节奏的状况，很可能是因为团队没有遵循敏捷的价值观，或是错误使用了敏捷实践。）

下面这些打断项目正常节奏的问题，是我亲眼所见的。

- 不知道先要完成哪些需求。
- 允许项目的需求收集阶段持续过长时间。
- 允许GUI随时发生变化，而GUI相关人员在项目中不知该如何是好。
- 没有架构的整体描述，不知道各个部分如何构成。
- 无法及时向项目中加入必要的人员，使得他们很难取得成功。

如果见到项目在努力寻找稳定的节奏，可以考虑使用下面的管理实践，也可以参见16.5节和第9章，从中得到更多建议。

8.2　举行中途回顾

寻找估算全部落空的原因

——史坦利，被人围攻的项目经理

我被难住了。不管针对什么，我们的估算总是落空，这个难题一直在困扰着我们。我们无法达成时间盒的要求。无论估算多么小，我们都无法达成。我以为这是个系统性的问题，所以决定在项目结束之前召开一次回顾会议，回过头看看过去几周的工作。

在这个中途回顾中，好几个人解释说：总是有市场人员和销售人员过来让他们“再新增一个功能，就一个”。我根本不用开口，因为技术带头人斯特拉说道：“要是我们在接受这些变更时不把它们加入到待办事项列表中，我们就永远无法完成之前承诺的工作。我们必须让史坦利看到这些新的需求。”

在回顾会议中做之后的行动规划时，团队想出一些方法让我接收到新的需求，包括让总是在叫着“再新增一个功能”的人到我的办公室。而我也意识到，我必须更频繁地发布产品待办事项，而不能光让人们去主动找到它。

如果没有这个中途回顾会议，我可能永远不知道为什么我们的估算总是会出问题。噢，也许到了项目后期会发现，可那时整个项目已经变成一场灾难了。按照目前的状况，我可以更加积极地管理项目后面的进展了。

要想了解项目中发生了什么，同时为未来的项目做规划，回顾活动是一种非常出色的方式。在项目结束时举行回顾挺好的，而且项目经理也可以把经验教训用到下个项目中。不过你也可以在项目的任何时候举办回顾，这样就可以知道明天是不是能换一种更好的工作方式。

敏捷式生命周期项目需要在每个迭代结束时举行回顾。顺序式、迭代式和增量式生命周期项目也要做回顾，不过是在达成主要里程碑之后。如果两个里程碑之间的时长超过一个月，项目经理可以举行一次中途回顾，以了解工作进展情况。

我推荐德比和拉尔森的书［DL06］，读者从中可以了解许多种举行中途回顾的操作方法。

中途回顾物有所值

项目经理皮特负责一个长达18个月的阶段–关卡式项目。原计划打算用6周收集一些需求，以开始架构原型化的工作。不过6周后，他们发现收集的需求根本不足以用来做原型化。皮特决定不再收集需求了（需求分析师要求说：“4周，再多给我们4周。”），他要举办一次中途回顾。

在回顾会议中，团队知道了为什么总是收集不到足够的需求。而且，他们也了解到：如果继续之前的工作方式，再给4周他们也收集不到“足够”的需求，至少还需要12周。项目开始6周后的这次回顾，避免了可能出现的3个月的延迟，而且也让项目团队相信：选择并坚持不同的生命周期模型势在必行。

8.3　为需求排序

我们有“超高”优先级的需求！

——帕翠莎，出类拔萃的业务分析师

公司来了一个新的产品经理汤米，他对工作充满热情。他讨厌对客户说“不”，承诺客户所有的功能都会在下个版本中提供。我总是搞不懂他的想法，他对这些需求总是只说一半，然后就再也没时间跟我细谈了。

压垮骆驼的最后一根稻草上周出现了。他决定我们的需求优先级不再划分为高、中、低三个等级，而是应该包括超高、至关重要、高、中、低这五个等级。

我跟他解释说我已经以最快的速度工作了。只要这些需求能够以数字的方式排序，我很愿意以任何顺序处理任何一个需求。“告诉我哪个需求是第1位的，我会先从它入手，然后赶紧交到开发人员手中，之后处理第2位的需求。”

他兴奋异常：“好主意！你看，所有的这些都是第1位的！”

“汤米，也许我没说清楚。你只能选择一个第1位的需求。只有一个。”

“可这样一来，我会让一些客户失望的。”

“我很遗憾，你向太多的客户做出了太多的承诺。但是我们只能有一个第1位的需求，一个第2位的需求。我相信只要你仔细想想，一定能找出来。”

> 果然，他做到了。他把所有的需求都排了顺序。接下来的一周，他想重新排序，不过我们已经开始处理第一部分需求了，所以他就没有打扰我们。这也是件好事，否则我就要干掉他了。（这句掐了，别播！）

幸运的话，项目经理可以把所有的需求都收集到一个地方，而且也知道先实现哪个，后实现哪个。不过据我所知，大部分项目经理面对的需求都散落在需求文档和缺陷跟踪系统之中，而且把它们按照高、中、低优先级进行排列。

接下来我要说的可能会让人觉得很悲观，可这是我在多个项目中亲眼所见的。项目中有太多的高优先级需求，以至于中优先级的需求根本无暇顾及，更甭提优先级低的需求了。这一大堆的高优先级需求也根本无法区分先做哪个。

到了下一个项目，提供需求的人知道你难于完成所有的高优先级需求，所以他们就想出来一个比高更高的优先级名称：至关重要。于是，很多需求的优先级就变成至关重要了。你可能还是要面对一些优先级为高和中的需求。可是现在，你连完成所有“至关重要”的需求的时间都没有了。怎么办？

给需求排序吧。从1开始，为每个需求分配唯一的数字。这样人们就能了解团队会什么时候处理哪些需求了。即使不按照功能逐个实现，你也可以按照排序的顺序来实现功能。

项目经理可以通过下列方法对需求排序。

两相对比。你可以把两个需求放在一起互相比较，然后问“这个需求的优先级比那个是高还是低？”判断出哪个高之后，它就成为第1位的了，相对较低的需求就是第2位的。

两相对比不太好操作，很可能会遇到判定项目驱动因素时同样的问题（参见1.4节）。项目经理要以温柔而坚决的方式进行，说明只能有一个第1位优先级的需求，也只能有一个第37位的需求。

按评判标准排定优先级。如果“两相对比”的方法没有进展，可能大家需要把需求的价值和优先级都明确表达出来，而这些一般是隐藏在他们平时的决策背后的。明确表述需求的价值，大家就可以根据这些值进行权衡，作出判断。

要想产生客观的条件，可以采用下列步骤：通过“头脑风暴”产生一系列可能的条件、将相似的条件分为一组、为每组分配一个分类名称。类别的名称就是大家用来评估的条件。我过去用过下面这些条件：

- 功能对架构的影响。

- 估算的实现时间。
- 该功能对于特别重要的客户的重要性。
- 特定人员实现或测试该功能的可行性。

为每个条件分配相对权重。负责主持头脑风暴的人可以问下面的问题：这个列表中，哪一个优先级最高？列表中是不是还有跟这一条优先级同样高的条目？这个列表中，哪一个优先级最低？列表中是不是还有跟这一条优先级同样低的条目？

知道了相对的顺序之后，就可以分配权重值了。优先级最低的条目权重值为1。每个优先级更高的条目，权重值为前一个的两倍。如果每种条件的优先级都是独一无二的，就会得到1、2、4、8这样的权重值序列。（你也可以定义高、中、低这样的优先级，而且每个优先级都可以有多种条件。所有低优先级的条件权重定义为1，所有中优先级的条件权重定义为2，所有高优先级的条件权重定义为4。）然后对应每种条件，为每个需求赋一个0～10的值，10说明是最重要的。请参见图8-1。

评判标准	权重	方法1		方法2	
		得分	权重得分	得分	权重得分
功能对架构的影响	1	10	10	3	3
估算实现时间	4	3	12	10	40
该功能对于特别重要的客户的重要性	8	1	8	10	80
特定人员实现或测试该功能的可行性	2	10	20	7	14
		--------	66	--------	137

图8-1　评判标准评估

制订并维护不断变化的需求日志：产品待办事项列表。如果有满坑满谷的需求，要想对它们都进行排序是非常困难的，我就遇到过这样的问题。要是有可能，只要一听到需求就应该对它们排序。可以将需求记录在电子表格或日志之中。如果没有使用敏捷项目生命周期，可以维护一个季度日志，记录这个季度要做什么，以及你认为接下来的三个或四个季度内要做什么。参见16.6.1

节，特别是那张季度待办事项列表的图。

8.4　用时间盒限定需求相关的工作

需求来了一茬又一茬，一茬又一茬……

——朗达，CIO

我的工作是：确保我们的项目可以完成，从而体现业务的价值。我们曾遇到一个拖了很长时间的重要项目，它原本预计6个月完成。项目开始2个月后，项目经理山姆跑过来跟我说：“我真快要把头发都揪掉了。需求总是来个没完，市场部的家伙们不停地添加新需求，而且也不说清楚现有的需求，我甚至都不知道该怎么启动项目。”

我知道怎么处理这个问题。我给市场部副总打了电话，跟他说他的人还有一周的时间可以用来定义需求。只要是定义完整的需求，我们就会做。如果只定义了一部分，我们就不管。就这么简单。

他试图搪塞我，告诉我他的人都不在办公室。于是我提议，如果一周不行的话，那干脆今天就停止需求收集活动。“别，我们会搞定的。”

我们没有开发他们要求的全部功能，但是确实做完了他们需要的功能。

如果项目中必须有明确的需求阶段，就用时间盒限制它。否则，需求分析和定义过程就有可能占用整个项目的时间。我曾参加过这样一个组织的工作，他们试图将项目的持续时间从18个月降低到6个月，但是需求阶段就用了4个月。在剩下的2个月里，他们没法做任何有用的事情。

在了解了一些情况之后，我知道了原因所在。市场人员过于忙碌，以至于没有时间与系统分析人员讨论，所以他们会推迟需求讨论会议。系统分析人员解释说他们还没有完成需求，因此希望从项目经理那里得到更多时间。这样一来，市场人员也就不急于解释清楚他们对需求的真实要求了。

项目经理决定把最初的需求获取工作限定为两周。市场人员因此满腹牢骚，在两周结束时，他们发出了好几份备忘录，说明为什么这个版本对于客户如此重要。不过开发人员们已经拿到了最重要的需求，可以开始开发产品了。三周之后，他们有了符合最初需求的产品原型，市场人员也已开始与系统分析人员合作定义后续的需求。

接下来，他们可以采用四周的迭代了。这样市场人员就可以用足够长的时间来定义任何给定的需求。不过项目团队知道，这些需求是要在下个迭代中实现的。

即使不适用迭代式开发，也要用时间盒限制初始的需求收集和分析活动。我是用下面这些方式完成的。

用时间盒限制初始的需求相关活动，并持续收集更多的需求。项目经理可以选择一个合理的短时间段（不超过项目总体持续时间的10%），要求收集到最重要的需求。风险在于，你可能得到了对客户来说是最重要的需求，但这些需求不一定能够决定最终的系统架构。

用时间盒限制所有的需求定义活动。这么做对于项目经理来说有点儿危险，可能会招惹来“你是一个没有团队精神的人”之类的废话。不过你可以这么解释：“我已经尽我所能，交付你们需要的产品。如果我允许需求收集工作还像上个项目那样耗费那么长时间，项目就不能及时交付了。如果你们觉得发布日期不重要，我们可以不这么干。可上一次，你们对于发布日期可是相当重视的啊。”

我在使用时间盒处理所有需求相关的工作时，项目团队会与需求相关人员一同收集、整理和细化需求。到时间盒结束时，团队实现了已经完成的需求，而且只实现已完成的需求。这种方式最适用于短期项目，而且是必须使用顺序式生命周期的项目。

提示：用一分为二的方式减少迭代持续时间

如果迭代效果不好，不妨试试把它一分为二。如果迭代持续6周，就改用3周。如果持续4周，就改用2周。如果是2周的迭代，就改用1周。更短的迭代有助于项目经理更快收集到反馈，洞悉团队在时间盒里的实际工作状况。

一旦知道人们在迭代中的具体工作情况，项目经理就能知道问题的出处了。是因为估算不准确，还是因为同时处理多个任务？也许是其他原因。项目经理就可以着手移除障碍。长期的迭代会掩盖问题。

8.5　将迭代限制在 4 周或是更少的时间内

我们完不成迭代的工作

——托弗，项目经理

这6个月来，我们一直试着用4周的时间盒推进工作，可我们怎么都完不成在迭代开始时估算可以完成的任务。

第一个迭代结束时，我们还剩几天的工作没完成，所以就把迭代扩展到了5周。下个迭代中的工作也相应增多，我们不得不把迭代增加到6周。后来就变成了8周一个迭代，而我们仍然无法完成为迭代规划的工作。

最后，我们决定采用一分为二的方式，也就是使用2周的迭代，看看能不能提高估算的准确性。结果发现，我们的估算没问题。可除了开发工作之外，我们还要做支持的工作，所以在一个迭代中，我们不能像预想的那样投入足够时间进行开发。我们的任务估算没有问题，但是对于工作时间的估算却落空了。

要是不采用更短的迭代，我们恐怕永远都不会发现是这么一回事。

也许你已经开始使用迭代了，可迭代持续时间是8周，甚至更多。现在你就该遇到其他问题了。在一个迭代周期中，测试人员无法完成他们的工作，要么就是开发人员总是过多估算自己可以完成的工作。或者，项目经理使用了螺旋式生命周期，但却没有按照完成一个完整功能的方式进行规划。

迭代时间越长，项目的节奏就越难保持。在顺序式生命周期中，项目经理可能已经见过类似问题，因为整个项目就像是一个迭代。如果很难保持项目的节奏，那就用更短的迭代吧，直到能够保持节奏为止。

这看起来好像违背直觉。“如果我使用6周的迭代都有问题，那4周的迭代又怎能帮到我呢？”答案在于：更频繁的反馈。迭代周期越短，要是项目可以因此达成特定的里程碑（理想状况下就是可发布的软件），那就很容易知道应该如何启动和结束一个迭代。“哈德逊湾式启动”（见4.2.4节）能够收到成效就是这个原因。如果不能保持项目的节奏，人们就很难进行准确的估算，他们会试着一次完成太多的工作，或者同时从事多个项目的工作，或者就不知道先从哪些工作入手，也许还会有其他特定于项目的原因。短时间盒会让问题暴露得更加明显，项目经理就可以着手解决。

8.6 使用波浪式的规划和日程安排

每次规划少量的工作，这让我更具灵活性

——唐纳德，项目经理

我们受困于顺序式生命周期。我们有严格的阶段关卡要求，而且管理层要审核每个关卡的完成状况。不过在各阶段关卡之间，我们使用时间盒，并对原型进行迭代，这样就能知道估算的质量如何。

现在我们处于设计阶段，而且大家认识到：我们无法让特定组件的性能符合要求。这是不可能的：它违反了物理规则。现在就得重新规划我们的工作了，发布的产品也会与规划的有些许不同。

我不担心日程安排。因为每次只规划4周的详细工作。我必须让大家知道接下来几周我对项目的规划，不过重新规划并不费事。我担心的是产品，不过那是另外一个问题了。

在项目中的某个点，某些风险的发生会改变整个项目的形势。软件项目会以不可预见的方式显现出本来面目。各个任务的完成时间也会发生变化。如果项目经理准备好用迭代方式反复规划，就可以提升项目日程安排的准确性。如果项目经理没有足够的产品相关知识，或是没有足够的历史数据来安排准确的项目日程，或是由于项目过大很难准确规划，那使用迭代式的规划和日程安排，就有助于项目应对技术风险和日程安排风险。

重新规划顺序式生命周期项目的时间安排。要先定义出顺序式生命周期的阶段或主要里程碑。确保团队明白选择的里程碑特定意义，要不就赶紧定义出里程碑的含义。比如，如果有个叫“需求冻结”的里程碑，就要明确说明它在项目中的含义何在。里程碑定义完成后，要为每个里程碑设定验收条件。这样一来，项目经理就可以知道里程碑何时算完成了。不要试图规划所有的工作。只规划将来3~4周的详细工作，这样每个人都知道接下来要做什么，后续几周要完成哪些任务。（如果你知道某些功能是要在项目的特定阶段完成的，比如演示、参加商业展览或是beta版发布，就得把这些特定阶段也定为里程碑。）

项目经理要做的就是监控项目进度，牢记里程碑验收条件。里程碑验收条件是为了达成特定里程碑必须要完成的任务。举例来说，我参加过一个团队，在顺序式生命周期中，他们将“需求冻结”作为里程碑，将“foo功能的需求完成并经过审核”作为验收条件。在顺序式生命周期中，人们很难达成“冻结”或是“完成”这样的条件，因为很难说早期的阶段是不是真地完成了，除非已经拿到了可以工作的代码，而且能够用它来表示需求或设计。

团队一完成手上的任务，项目经理就可以向当前的详细规划中添加新的任务了。假定项目经理制订的是4周的波浪式规划，如果目前处于第3周，那就可以规划第7周的工作了。想通过迭代的方式来规划顺序式生命周期项目的工作比较困难，因为进度难以评估，除非已经接近某个里程碑了。这就是为什么需要里程碑验收条件的原因。

使用波浪式规划的顺序式生命周期有个问题。由于项目分成各个阶段（需求、分析、编码，诸如此类），在项目开始时，人们很容易倾向于规划尽量详细的工作。不要规划所有的细节，只为接下来的几周规划详细工作就可以了。

项目完成一个前面的阶段后，就可以为后面的阶段准备更多细节了。随着项目进展，看看是否存在技术债务的状况。如果后续阶段所费时间超出预期，这就是存在技术债务的警讯。技术债务越多，后面阶段所耗费的时间越长。

如果顺利进入编码和测试阶段，项目经理就可以规划最后的测试和项目临近结束的活动了。比如早期版本发布、beta发布，还可以有其他在最终发布之前的活动。

项目经理已经使用波浪式规划来构建项目日程，那么在顺序式生命周期中，项目经理要创

建里程碑，在完成一个阶段后要重新规划这些里程碑。要将重新规划活动放入最开始的项目日程安排中。利用目前对项目的了解（特别是在项目中期回顾得到的信息），更新项目的规划和日程安排。

这样一来，团队就能知道项目是受控的，团队和项目经理可以不断评估并管理与日程安排相关的风险。

提示：将重新规划活动放到项目日程安排中

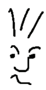
要是没有使用敏捷生命周期组织项目，还是将重新规划活动明确标识出来吧。还要经常重新规划，这样项目日程就不会在众人领会它之前脱离现实了。

为迭代式、增量式和敏捷生命周期进行重新规划。由于这几种生命周期模型都包含迭代或增量式活动（或者两者皆备），相比顺序式生命周期，要进行重新规划就简单多了。

对于每次迭代不会将完成的代码交付到代码库中的迭代式生命周期，可以使用与顺序式生命周期相同的方式进行重新规划。可能会有不同的里程碑，比如“探索原型1并发布结果”，不过背后的理念都是一样的。

在项目启动时，增量式生命周期进行重新规划类似于顺序式生命周期，因为一开始时也是倾向于分阶段的。一旦增量开始后，项目经理会发现，要制定波浪式的日程安排很容易，因为不必等到一个阶段结束。你所期待的是将完成的（即经过开发和测试的）代码签入到代码库中。

敏捷生命周期会自然而然地使用波浪式规划，因为每个迭代都是一个时间盒。项目经理规划的工作只需启动迭代即可，要在迭代中监控进度，而且确保迭代完成时提交完成的工作。



小乔爱问……

是不是迭代式日程安排不可避免?

不是每个人、每个项目都适用迭代式日程安排。它最适合用在下列这些情形下:

- 当你知道要做什么，却不知道应该怎么做的时候。
- 当你面临时间压力，而且希望利用项目现有进展的时候。

这样以来，研究型项目或是项目的研究阶段就不在适应范围之中了。要在研究型项目中使用迭代式日程安排，就得制订出每个时间盒结束时需要提出的问题。然后就可以用迭代式日程安排来重新规划下一个时间盒的工作了。项目产出的不是产品，而是对于一些问题的答案——或是更多问题，这是一种不同的交付产品。

8

8.7　创建跨职能团队

各自为战使我们一叶障目，不见泰山

——布莱恩，开发经理

我是一名开发经理，负责一个大型的事务处理产品。我们有一个GUI的前端，一堆中间件，后端是为数众多的数据库。

所有的开发人员在一起工作，设计和开发产品。我负责管理项目。然后我们就会把开发的程序交给测试人员。测试经理南希会负责发布产品。

完成开发后，我们会开始下一个项目的工作。同时我们会根据测试人员对之前项目返回的测试结果进行修复。这对当前项目的开发工作形成很多干扰。而且，客户开始使用产品后也会发出抱怨，他们会发现开发人员和测试人员都没有找到的问题。

为了以后的项目考虑，南希和我坐下来谈了谈。我们该做些什么，才能保证完成现在的项目，并且不受所有这些干扰的影响呢？我们决定将测试人员加入到开发团队中。南希承担项目经理的职责。对于跟踪细节，她比我在行。

结果真令人惊喜！开发人员喜欢跟测试人员一起工作，而测试人员也十分愿意跟开发人员一起工作。在设计时，测试人员能够发现开发人员想不到的问题。而有些开发人员也喜欢炫耀他们精巧的想法，帮助测试人员测试。后来，等到产品上线后，几乎没有任何来自客户的抱怨，我们也可以把全部的时间都用来应对新产品了。

我们不再把工作在各人之间传来传去了，大家不再各自为战，我们现在以跨职能团队的形式工作。

在第7章中，我建议项目经理应该在自己的团队中配备几种不同角色的人员。团队的构成应该是跨职能的。跨职能团队有以下好处。

- 跨职能团队的工作效率更高［Mey93］。单一职能团队可以更快地完成各人负责的部分，但是没有人可以复查或验证他们的工作质量。整个产品也不能早日完成。真正能让项目团队得到认可的，不是因为完成了需求，而是因为实现了**全部**功能。
- 跨职能团队具有多样性。测试人员试图从可测试性方面考虑问题，技术文案跟开发人员和测试人员一起确认如何表示项目产品的工作方式，分析人员不断细化需求，并有可能参与构建验收测试。

8.8　根据项目的风险选择生命周期模型

标准流程对我们来说不管用！

——辛西娅，项目经理

我们属于一个大公司，称我们为研究所，也算是八九不离十。公司有一个项目管理办公室（PMO），他们会定义所有的项目管理流程。

几年前，他们定义出了“标准的”项目流程。一个标准的项目是很庞大的，通常超过100个人，而且会持续数年之久。当然，没有哪个项目经理能够按照PMO定义的方式来成功管理一个标准的项目。

我们决定通过不同的方式来管理项目。我们很聪明，可以产生PMO需要的文档和度量数据，而且仍然使用我们自己的方式来管理项目，以取得成功。我们的项目规模大小不同，时间长短不同，有些项目的技术风险很高，有些基本上没有任何技术风险。而且，当我们必须要管理成本的时候，我们也能做得到。

现在，即使是PMO也说：“根据你们的风险选择生命周期模型吧。选择哪些实践，也要考虑到你们的团队和所选生命周期模型的现实情况。”我们一直没有失败，部分原因在于不再去碰那些无比庞大的项目。我们从小起步，保持小规模，使用迭代、时间盒和增量式开发，因为它们都很有效。而且我们与团队一起，基于实际情况，选择符合我们要求、能够为我们所用的实践。

对我帮助最大的，就是认识到：我们不一定非得用顺序式生命周期来管理所有的项目。我有其他选择！一旦选定了生命周期模型，再考虑用哪些实践就很容易了。

在第3章中，可以看到各种生命周期模型能够明确展现和解决的风险。选用何种生命周期模型，是项目经理必须先决定的几件事之一，而且对如何组织项目会有长时间的影响。要三思而后行。如果不能确定，不妨先使用敏捷生命周期，因为它能尽早带给项目最大的灵活度。

8.9　保持合理的工作时间

每天晚上都在公司吃晚餐毫无道理，对任何人都是如此

——贾斯汀，开发总监

我们大概有150人投入了下一个版本的开发，当然，这个版本对公司的发展至关重要。整个进度陷入了停滞阶段。我的老板——公司副总说现在到了每个人都得在公司吃晚餐的时候了。

计划原本是这样的：每晚7点，有一名总监会负责安排所有参与该项目人员的晚饭。我们聚在一起，共进晚餐，之后大家会继续工作，努力多干些活儿。

头一周看起来还不错。接下来，有些人就得上午10点或11点才能到公司了。有一个人甚至下午才能到。而且，不少人吃完晚餐后就离开了。我整天在办公室里面转悠，看到有人在整理私人账目，有人在给伴侣或是父母打电话。还有一位，正在帮自己的孩子安排周末聚会时间。

最后，我们停止了这样的做法，恢复到每个人每周工作40小时。突然间，代码和测试的质量开始提升了！我们也能完成更多的工作。

项目经理在面临很多问题的时候，会很容易产生让团队一起加班工作的想法。可是如果人们加班加得越多，他们能完成的工作反而越少［DL99］。为什么？请往下看。

人们每天最多只能完成6个小时的技术工作。在较短的时间段内，至多1~2周，有些人也许每天能多干1~2个小时。可是大多数人都不能进行长时间的加班。

长时间的加班会扰乱一个人每天的节奏，会让他把更多时间花在喝咖啡、吃午餐、给朋友或是父母打电话、浏览网页、整理私人账目、关注自己的状态等与工作无关的事情之上。要不了多久，这些人的心不在焉就会在他们的项目工作产出上反映出来。

项目经理如果发现项目工作进展过慢，可以考虑下面提到的这些方法，并用之来维持项目的节奏。

8.10　使用“小石子”

乔无法估算超过一周的工作

——艾德里安，开发经理

我的团队中有一个技术高手——乔。他的工作非常杰出，去年由他设计的一个架构表现得很出色。乔可以预估小量的工作，但是他不善于估算大任务。不管是大项目还是任务，如果要他来估算，总会出现偏差，低一个数量级，甚至更多。

在上一个项目中，乔和他的小组负责一大块任务，老大一块儿。因为任务过大，乔无法从全局把握。他只能估算自己能够看到的部分。可是有些任务就像冰山一样，乍一看，你所能观察到的只是任务的冰山一角。只有深入进去，才能了解任务的全貌。

现在我们会把每个任务都拆成“小石子”（有时会用“探索式开发”的方式）。乔现在的估算就好多了。我也不再给每个人的估算加上一个随意的经验系数了。

如果要耗费很长时间才能完成一个任务，项目就会丧失节奏。团队成员看不到任务结束的时候，他们就会变得懈怠，从而没有节奏感。“小石子”可以帮助每个人保持自己的节奏，并让项目不至偏离正轨。

提示：帮助团队成员避免“学生综合症”

如果人们直到最后一刻——有时甚至已经过了这个时刻——才开始某个任务的工作，这就发生了“学生综合症”[Gol97]。如果人们用周来估算任务，就很容易引发“学生综合症”，用天或是“小石子”的方式就不会。

“学生综合症”会导致项目延迟，打乱项目节奏。如果汤姆无法完成自己的工作，而杰瑞又得等汤姆工作完成后才能开始，那杰瑞现在就处于“等待”状态了。

通过指导团队成员使用“小石子”估算任务，或是使用有时间盒限制的迭代，就可以控制“学生综合症”。不管哪种方式，每个人每天要有所交付，可以有效避免“学生综合症”的发生。提交自己负责的工作所产生的一点点压力会让很多人开动起来，不再让别人空等许久，他们不想拆同事的台。如果人们能看到自己的工作成果，他们就会继续不断取得进展。这样一来，每个人都能保证合理的工作时间。

8.11　管理干扰

干扰是我们的障碍

——乔希，工程副总

我一直伴随着这家公司成长。当初我们只有7个程序员，十年之后，现在已经发展到几百名开发人员、测试人员、文案、发布经理和其他各色人等。我花了一些时间才认识到：对我们来说，干扰是个很大的问题，特别是我们还是一个小公司的时候。

我很早就知道干扰是个问题，不过却没有认识到它的严重性，直到项目经理泰德给我一个列表，罗列出了他的团队在过去一周遇到的种种干扰。为解决这个问题，我采取了如下措施：管理项目组合，使用有时间盒限制的迭代，将更多注意力放在我们如何向组织和项目中引入新人之上。

虽然我们现在还没有做到足够完美，但是干扰不再对项目产生严重负面影响。

干扰会摧毁一个项目的节奏。一次干扰还好说，要是所有的干扰一起上阵，就像被一群鸭子围攻，也能致人死地。干扰会导致人们丧失多达40%的时间［RG05］。有两种类型的干扰：其他项目和其他人。

8.11.1　应对其他项目干扰

作为项目经理，你应该保护迭代的工作。如果使用顺序式生命周期，整个项目就是一个迭代。如果使用时间盒，迭代就是时间盒的持续时间。在迭代式生命周期中，迭代持续时间可以发生变化，不过与要完成的工作有关。在增量式生命周期中，本质上是没有迭代的。开发某个功能的时间也许可称之为“迭代”。

要推迟来自其他项目对团队的干扰，除非当前的迭代结束。结束后，可以推迟下个迭代的开始，处理干扰。

人们有时不知道他们的干扰会影响你的项目。将一周之内发生的干扰记录下来，让人们看到这些干扰的影响。要以事实为依据，但是不要去责怪别人——项目经理要起到教育和通告的职责。

8.11.2　应对其他人的干扰

项目经理要建议团队成员及时互相讨论项目相关的问题，项目工作会因此而取得进展。不过当有人向别人提问时，被提问的人就相当于被干扰了。在充满隔间的办公室中，被提问者周围的人也会受到干扰。项目经理该怎么办？

你有下面几个选择。我的首选建议是：鼓励大家结对编程（或是结对制订需求、结对测试），这样人们就可以共同学习和了解系统。如果结对不起作用，就要让人们在隐秘的空间中讨论，从而不至影响其他人。如果很难获得一个“隐秘”的办公室，就弄一个项目“作战室”吧，可以在其中贴出项目的仪表板、架构文档等各种产出物。

要保证有配置完备的电脑，人们可以用其访问代码库或是其他电子文档，共同讨论同一个产出物。

如果从未要求过“家具警察”［DL99］的帮助，项目经理可能会觉得有点儿难。不过“家具警察”也是人。看看下一节中的文本框，想想能做些什么。我发现用“点心加啤酒[①]”这样的小恩小惠效果不错（不过不是总能对同样的人起作用）。

8.12　管理缺陷，从项目初就开始

缺陷？我们没有什么烦人的缺陷！

——爱德华，程序经理

我给你讲个故事吧，这还是在我开始非常重视缺陷之前。那时我对软件开发还不甚了解，虽然已经管理过不少项目，不过绝大部分都是销售项目。开发人员完成编码工作就高高兴兴下班了，而测试人员就一直在抱怨他们无法让软件通过构建过程。我去询问开发人员，他们每个人都会说：“在我的机器上没问题啊。”我觉得这帮搞测试的也太孩子气了。

我们的一位资深经理生病了，而且要休长期病假。由于我是唯一有过管理公司经验的人，因而资深管理的角色就落在了我的头上。项目本来预计再有2个月就可以完成，所以我就找了咨询师珍妮丝来管理项目。

珍妮丝所做的第一件事情就是记录缺陷列表。她甚至强迫开发人员使用缺陷跟踪系统。人们都来向我抱怨，所以我就去问珍妮丝那个系统是怎么回事。她解释说，开发人员从项目一开始就置缺陷于不顾。即使他们想起来了，也只是会想到刚修复的缺陷，而不是还没有解决的缺陷。珍妮丝说：“如果不着力解决这些缺陷，这个项目永远都完不成。实际上，还有一个开发

---

① 要注意组织对于日常工作日饮酒的政策规定。很少有公司会在甜食方面做出限制。

人员跟我说：‘缺陷？我们没有什么烦人的缺陷！’”然后珍妮丝告诉我，她认为我们已经有数百个没有解决的缺陷。“这些缺陷让我们的测试不能继续，而且客户也因此而不能使用系统。老实说，我觉得缺陷已经让一些开发人员没法继续写代码了。”

珍妮丝和我又讨论了一会儿。她说服我这是一个很大的问题。她是对的，项目当时累积的缺陷使得我们无法达成最初2个月的截止期限。我们又用了4个月才发布系统。不过我们做得不错，客户也很满意。现在我再也不会将缺陷置之脑后了。

如何影响别人

只要你愿意放弃“命令和控制”式管理所带来的幻象，学习影响别人就很容易了。也许需要你在心态上做出调整。下面的提示供你参考。

- 记住问题不是你一个人的。项目经理有时会认为自己必须提供所有的建议和解决方案。并非如此。在项目中，发布日期、功能集合、缺陷水平，每个人都对这些有责任。不要认为一个问题就是你自己的。你有责任让团队解决问题，而不是强令他们如何解决。
- 思考你能为组织带来的价值。一旦明白了自己的价值，你就可以想想这些价值对于别人的意义了［CB91］。这些价值可以帮你寻求别人的帮助，并给予回报。
- 发现其他人或团队的驱动力（WIIFM，what’s in it for me?）。发现其他人或团队中为我所用的东西。有人喜欢做有意思的事情，有人希望得到公众或是个人的认可。很多人希望把工作做得出色，而且知道自己能为整个项目的成功做出哪些贡献。大多数情况下，参与多地点项目的团队有其特定的激励因素。要把这个因素找出来。
- 建议项目经理和负责的人（或团队）共同面对问题。这可以让项目经理能够友善、开放地接受他人的建议和想法。如果你直接告诉别人怎么做，他们会有千千万万个理由告诉你“你错了”。如果让他们自己找答案，他们会更容易去解决问题。
- 倾听团队。团队会告诉项目经理他们要怎么样才能达到最高的工作效率。如果你让人们变换工作方式，他们可能会需要不同的工作空间、更多设备或是其他东西。
- 在讨论时，给他人思考的时间。在推行你自己的想法时，要允许别人认真思考和质疑这些想法。有些人可能需要多一点时间来提供有价值的反馈。
- 不要固执己见。如果你借助影响力，以协作方式工作，别人就能改善你提出的解决方案。要确定你不会影响他们。我们有时会发现很难抛弃过去卓有成效的想法。要记住，你可以把自己的想法作为解决某个问题的起点。

在项目进行过程中，很多团队对缺陷都采取放任自流的态度。直到项目快结束了，他们才会认真考虑如何管理缺陷。如果使用顺序式生命周期，在项目开始时可能不会有多少编码缺陷等待审核。



小乔爱问……

如果运营与开发同时进行，我该如何应对干扰？

这与选择何种生命周期模型无关。如果你正在研发新产品，而运营相关的工作不断出现，这就说明你的估算有问题。（在提供支持和开发工作同时进行时，也会发生同样问题。）来自运营的干扰会毁掉你的项目。

可以考虑下列建议。

- 调出几个开发人员，用1~2周的时间专门负责运营工作。并且在运营期间调换这些人员。
- 假定每个人每周只能用2~3天进行开发，其余的时间都用来处理突发任务。每个人在一天之内不能从事多个任务。
- 加入更多人手，这些人必须是喜欢从事“灭火”任务的。他们的首要职责就是解决突发问题，接下来才是项目中的任务。
- 成立一个小组，组员的职责就是负责运营。
- 如果项目经理使用迭代，并以相对大小和持续时间进行估算，不妨估算每项运营工作，然后将其加入到产品的待办事项列表之中。

不管选择何种方式，你都要对运营工作和开发工作负责。

如果项目经理不在项目初期就开始管理缺陷，那缺陷就会反过来控制你。项目的技术债务（参见附录B）会不断增长，项目经理不到最后根本无从知晓。到最后就会有太多缺陷需要修复，也就没法全部搞定了。

项目经理有下列选择。如果条件允许，不妨转而使用敏捷生命周期，开发人员可以与测试人员同时进行开发和测试。团队报告的总体缺陷数目会减少，项目经理也能更快了解到缺陷，并且马上着手处理。

要是不能使用敏捷生命周期，就用增量式生命周期吧，还要确保开发人员随工作进展进行持续集成（参见9.1节）。让测试人员用不同方式来测试开发人员已经完成的功能。

在缺陷跟踪系统中组织缺陷时，要考虑如何对缺陷分类。可以为每个缺陷定义严重程度和优先级。而且，项目经理也不要对后来出现的缺陷玩“缺陷提升和降级的游戏”（参见15.4.2节）。严重程度属于问题的技术层面。如果严重程度很高，系统就无法运行，要么就会提供错误结果。优先级属于问题的业务影响。如果优先级很高，客户会受到问题的严重影响。

在评估缺陷时，有些团队使用类似风险分析表的表格。图8-2中的缺陷并不是一个很大的技术问题（在严重程度上比较低），但是优先级比较高，因为很容易造成混淆。

缺陷序号	简要描述	优先级	严重程度	暴露程度	何时修复?
17	姓名与地址混淆	高	低	高（客户会感到迷惑不解！）	当前迭代

图8-2　缺陷评估表

表中“何时修复？”字段，可能存在缺陷跟踪系统中。我见过团队使用日期、发布版本、迭代填充该字段，都取得了很好的效果。



铭记在心

- 作为项目经理，你要带头考虑使用或调整哪些管理实践。
- 评估项目的问题，然后根据这些问题来判断使用或调整哪些实践。
- 要寻找可以建立和维持项目节奏的实践。

8

第9章 保持项目节奏

9

项目经理不但要用管理实践掌控项目，还可以鼓励团队改变自己的技术实践，从而获得更大收益。本章包含的一系列实践，能为项目带来很多好处。项目经理和团队要根据自身的实际情况，判断如何调整、使用这些实践，而不要强制推行。如果你认为它们有所裨益，不妨将其介绍给团队，并欢迎团队积极尝试。

9.1 在项目中使用持续集成

开发人员用一两个小时编写一段代码，对其进行编译，测试，复查，构建，运行冒烟测试，并验证做出的改变不会破坏现有系统代码，再将其签入到代码库中。持续集成就这么发生了。[①]

持续集成可以让开发人员马上得到所做工作的反馈。他们开始考虑将大任务拆分成更小的部分（这可以让他们以更快的步伐前进），还能尽早识别出项目的集成风险。持续集成让开发人员每天都能有所收获，帮助项目团队找到符合自己的节奏。

开发人员只有在完成了某个功能的代码后才将其签入，这就是按阶段集成的方式。有些开发人员一周签入一次代码。有些更糟糕的项目，开发人员一到两个月才签入一次代码。按阶段集成会打断项目的节奏。构建时出现的问题，会中断设计和编写新功能代码的人们的思路。他们不得不记住很多小细节，而这些小细节覆盖了从项目开始直到进行集成的各个阶段。如果忘记了这些细节，项目进度会放缓，因为团队发现了缺陷，而且必须想起可能是几个月前所写代码的细枝末节。这其实是一种同时处理多任务的方式，而且特别不容易发现。人们是在做同一个项目的工作，而且手上的任务可能也相关，但是思考问题时的抽象层次却有所不同（设计比调试的抽象层次更高），而且也不再是处理同一个功能的问题，思考问题的上下文由此发生了变化。所有切换上下文的代价都发生在这里（参见16.7.3节），在设计下一个功能时，开发人员可能会因此而丧失好的想法，还可能增加向设计中引入缺陷的潜在风险。

---

① 见http://www.martinfowler.com/articles/continuousIntegration.html。

频繁构建是为开发人员和项目经理准备的

如果你提出使用每日构建，测试人员可能会抱怨：“我们做不到以如此快的速度构建，只能每周构建一次。”频繁构建，比如每小时甚或每天，不是为测试人员准备的。频繁构建可以为开发人员提供反馈，同时让项目经理了解项目状态。

如果测试人员可以利用每日构建来运行测试，那就太棒了。不过即使测试人员做不到这一点，通过每天构建可工作的代码，并维持这样的节奏，开发人员也可以从中获得有价值的反馈。

如果项目经理不能将变更后的代码集成到代码库中，你可以采取持续集成的变种。假设你管理一个项目团队，大家的任务是扩展一个已存在产品的设计。团队目前正在处理的部分产品代码是整个产品的基础。如果这部分代码无法运行，那整个产品都不能运行。你现在不想签入代码，因为这样会破坏现有的产品，而且会占用三个月的时间来添加和替换产品的现有功能。你该怎么做？

应该选择现有状况下的最佳方案。你可以将主代码库做一个分支版本，并让所有开发人员在这个分支版本上开发。他们每次签入一些代码，都会与主线代码进行同步，保证自己的分支代码是最新版本。这使得开发人员总是在最新和最全的代码版本库上进行开发，而且他们一直都在进行集成。当他们完成各自工作后，就会将所有代码合并回主线。

这是持续集成的一个变种。如果必须重写已经存在的代码，而且在开发变更代码时不想破坏现有系统，可以采用这种方式。如果项目经理管理的项目所提供的服务要为一组产品使用，比如一个程序库或是一个平台，这种方式同样适用。

9.2　为构建创建自动化冒烟测试

不管使用持续集成与否，都应该为构建产品创建自动化冒烟测试。冒烟测试仅仅是为了验证要构建的版本没有问题。你想添加多少回归测试都可以，我不拦着，但是使用冒烟测试，是要知道当前的构建版本是否对大家有用。

有了自动化冒烟测试，项目团队就可以知道是不是有人破坏了当前构建版本。如果能尽早知道一个构建完成了，你就可以在其基础上做些自己想做的工作。要是必须依赖其他开发人员或是测试人员才能知道当前构建版本出了问题，那你就不能以最快的速度修复当前构建版本了。

别让烟跑出去！

——梅瑞狄斯，资深测试人员

作为测试人员，寻找问题是我的职责所在。而且，我也很擅长这个。我有一句座右铭：

“别让烟跑出去。”

有一次，我加入了公司的一个新项目。开发人员以前从没有与专业测试人员一起工作过。我拿着一张有15个缺陷的列表，走到他们中一个人的面前，然后说：“你得说说这是怎么回事，要不我把这些问题提交到缺陷跟踪系统里面如何？”他们都惊呆了。

我向他们说明，这些缺陷通过一个自动化冒烟测试都可以找出来。没有哪个缺陷需要靠我的专业知识和经验才能发现。我不想浪费时间找出这些易于发现的缺陷，而是要找出更捣蛋的缺陷——时而发生的问题、系统可伸缩性上的问题和可靠性问题。我想钻到代码之中，把高难度缺陷一个个都揪出来，直到它们全部显形。

那个开发人员脸色发白，屏住呼吸说道：“呃，好吧，我希望你也能那么做。需要我配合你什么？”

我解释了我的座右铭：“你的工作是尽量保证不出问题。我的职责是尽量让问题血淋淋地暴露出来。明白了吗？”他表示同意。我想我听到他跟别人说我有点嗜血倾向。不过，管他的，我对此毫无意见。

我签入了我的测试代码，开始构建自动化冒烟测试框架。开发人员开始一点儿一点儿地加入冒烟测试。如果我找到了一个冒烟测试应该发现的问题，就会冲到加入这个缺陷的开发人员跟前，对他说“别让烟跑出去”。不久之后，大家只要一看到我走向开发人员的小隔间，就会有人大叫：“是谁把烟放出去的？”

我们现在合作得非常好。这些开发人员的工作表现非常好，而且他们也让我能很好地完成自己的工作。我们的产品NB极了！（这么说没事吧？）

让构建版本正常工作，这对建立并保持项目的节奏大有裨益。能够第一时间发现构建版本出现问题，也能帮助项目经理将项目带回正常的节奏，或者发现是什么使得团队无法保持节奏。

实际上，如果项目的产品要在多种型号的计算机、数据库或固件上运行，就要确保团队每天都为所有的平台编译并构建产品。如果不这么做，到了项目结束的时候，项目经理就得忙着解决那时才能发现的问题了。在项目中越早发现兼容性方面的差池，就越容易应对，而且开发人员在后面的工作中也能将其牢记在心。

9.3　按功能实现，而不是按架构

逐个功能进行实现和测试

——哈威、维贾、道、兰迪、肯和梅布尔，负责提醒功能的团队

我是哈威，团队的头儿。我们的开发人员包括维贾、道、兰迪和肯，梅布尔是测试人员。

我们在GUI、平台、硬件集成等方面都有专家，而梅布尔无所不知。（背景中传来笑声。）

刚开始的时候，我们试图先把架构开发出来，再制订规范，告诉大家我们在做什么。我们每个人各自负责一块，最后再放到一起。可是一切都没有按计划进行。因为即使我们有规范，在开发各自的组件时，我们总会改变些什么。

梅布尔曾经听过有关按功能实现的说法，并告诉了我。我问大家是否愿意组建一个基于功能的团队。嗯，其实是基于功能集合的团队。我们开发所有的提醒功能，一次完成一个，从前端到后台。梅布尔负责测试。

开始时有点儿困难，不过习惯之后，我们添加功能的速度让人惊喜不已。我们不再是每个人开发很多代码然后再集成到一起，而是只添加每个提醒功能需要的代码，再进行测试。梅布尔保证我们不要出现系统级别上的问题，我们保证每个提醒功能不要出问题。

我们的开发速度如此之快，不久就有新的功能需要开发了。现在，有些其他组也在尝试我们的做法。整个项目的进度都加快了，而且客户也能知道我们在开发哪些功能，反馈也由此而来。

很多项目团队都是按照架构进行组织的，比如基于Web的产品会分平台组、中间件组和GUI组。可是如果按架构进行实现，项目经理就很难知道开发完的功能是否可以正常运行，除非等到整个产品集成完毕。如果按架构实现，就很难进行持续集成，因为无法构建和运行测试。在开发的开始阶段，没有哪个功能是完全实现的，都要等到快结束的时候才能做完。而且，项目经理也无法将任何功能视为完成。

所有已经完成的开发工作都是在创建“浪费”［MP06］，没有产生任何有价值的东西。一旦完成某些功能，就可以开始计算价值了。可是在此之前，这些价值无法衡量。

按架构实现会打乱项目的节奏，因为得到的都是部分完成的功能。不到开发完成，项目经理看不到完整的功能，而这时得到的反馈对开发人员来说就太晚了。如果按功能实现，开发团队只要针对某个功能的要求利用架构进行实现就好了。假如要开发Web应用，项目经理可以组织一个小型的基于功能的跨职能团队，其中包括一些了解平台的人员、一些了解中间件的人员和了解GUI的人员，他们各自负责自己擅长的工作，并且只针对团队负责的一个特定功能。要是团队人员拥有不同的兴趣和技术技能，比如擅长GUI和固件方面的技术，这些人就可以按照功能逐个应用他们的技能。（项目经理应该帮助他们组织好各自在项目中的工作时间。）

有些团队认为，应该在项目前期预先细化需求（Big Requirements Up Front，BRUF），并进行大量设计（Big Design Up Front，BDUF）。这样的团队要想切换到按功能实现的方式，就没那么顺利了。这时，可以跟他们说明一个功能应该是什么样子（它应该有多小），并让他们知道自己不应该做什么（参见5.3节），帮他们了解实现的功能要具备足够的架构完整性，从而在实现后

续功能时也不会有问题。要提醒这些团队，他们在调试和测试的时候都是按功能进行的，因此不会对此陌生。

架构反映组织结构

你可能已经注意到，产品的架构反映了团队的组织结构。产品的规模越大、复杂性越高，这一点就越明显。这就是康威定律（Conway’s law［Bro95］)：任何软件都会反映出制造它的组织结构。

如果团队更倾向于以架构为导向，产品中包含的组件就越多。这会阻止人们协同工作，并减缓项目进度。他们不仅难以共同完成完整的功能，而且随着工作的进行，他们也丧失了重构系统使之具备更高内聚性的机会。机构的组织方式阻止了他们这样做。

如果你看到只有一两个人负责的非常短小的代码，或是发现一大堆耦合在一起或内聚性很差的代码，那就仔细观察一下组织架构吧。这两种极端情况都会打乱项目节奏，实际上，项目可能已经没有节奏了。令人惊讶的是，通过调整组织结构，项目的问题也可以得到解决。

有些人可能仍持反对态度，说他们在考虑一个功能之前，仍需要知道整体架构没有问题。这种情况下，架构草稿会有所帮助（参见3.5节）。不过不要坚守其中的架构，除非团队已经实现了不少功能。

9.3.1　首先实现具有最高价值的功能

在按功能实现时，要先实现最有价值的功能，把风险最高的功能往后放一放。如果运气足够好，也许根本不用开发这些高风险的功能。这样一来，测试人员和开发人员就能对整个系统有足够的了解，就可以保持自己的节奏了。

有些团队不愿意按功能实现，因为他们不知道先开发哪个功能。团队中有些人会希望先开发风险最高的功能，有些人希望先开发最有价值的功能。如果没人愿意排定需求的优先级顺序，那就很难知道哪个功能最有价值了。

如果没有客户或是客户代理的参与，作为项目经理，你就要在团队的辅助下，担起制订和发布产品待办事项列表的责任（参见16.6.1节）。项目经理要判断出各个功能的实现顺序。

如果有人愿意判定各个功能的价值，那就按价值高低进行实现，即便这会给架构带来风险也要这么做。

功能的价值和完成度越高，项目经理就能在当前项目中为自己和团队带来更多灵活性。你也许可以尽早发布（如果高风险的功能无法使用当前架构），这对很多客户都是很有价值的。推迟开发高风险、低价值的功能，可以保持项目的节奏，并降低当前版本的风险。

9.3.2 按功能调试

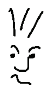
有些团队坚持按架构实现。到了测试的时候，测试人员却是依逐个功能进行测试的。这么一来，团队在调试的时候也得按功能进行了，即使他们构建产品的方式与此不同。

为了按功能调试，项目经理要构建跨架构团队（要么就得能够接触到所有团队成员）。如果从来没有组建过跨越架构的团队，项目的节奏就会因此而扰乱。不过，要是没有跨越架构的团队，问题是无法解决的。

要是项目到了尾声，发现团队是在按功能调试，那还是考虑在一开始就按功能实现吧。这会减少项目的干扰，同时项目节奏也更平稳。

9.3.3 按功能测试

测试人员会按照功能测试系统，有时是因为需求的编写方式，有时是因为这就是测试人员访问系统的工作方式。当测试人员报告问题时，他们很少会单独报告各个小架构组件上的问题。实际上，他们会在测试用例中提出问题，比如，“当我试图打开一个银行账号时，我可以看到数据穿过中间件进入到数据库中，但是我看不到返回的确认。”测试人员在这里报告了一个中间件的问题，但是没有明确指出是在哪里。



小乔爱问……

如果产品中包括硬件部分，我该怎么按功能实现？

如果产品中包括硬件，也许就很难完全做到从项目一开始就按功能实现了。不过你还是可以像下面说的这样做。

当硬件团队完成设计的时候，要准备对原型进行迭代。也许可以做到按部分功能进行实现，用存根代码（stub）取代实际的硬件实现。不过要小心，这时开发的任何代码都只是原型，不是最终的代码。如果硬件团队能够提供一个模拟器或是仿真器，那代码可能就与最后版本没多大差别了。

硬件团队开始时可能雄心勃勃，到最后真正完成的功能却没有那么多。除非硬件已经成型了，你无法知道他们是否能够实现当初规划的所有功能，也不知道他们对于软件设计方面的反馈能否及时。还是准备对原型进行迭代吧。

如果不必担心软件开发方面的成本，你可以尽所能把这些原型做到最好。很多硬件和软件结合的项目，对于软件开发的成本都是不必担心的。如果你对此确实有所关注，那就要保证不要在软件原型上花费过多不必要的时间。

一旦硬件有了初步的物理实现，你对于硬件的功能也就有了更明确的想法，也就更清楚软件应该完成哪些功能了。此时就可以按功能逐个实现。尽量少实现每个功能需要的软硬件交互接口部分，这样需要调试的部分也就减少了，而且要是出现缺陷，也能很快找到源头。

开发人员已经习惯于看到这样的报告了（不管是与调试还是测试相关），也惯于回溯问题，以判断功能是如何与架构交互的，从而理解问题。按功能实现，开发人员就可以在设计和编写代码时看到潜在的问题，不至于到开发尾声才发现。

9.4　多几只眼睛盯着产品

邀请团队成员相互复查工作。复查的形式并不重要，结对编程（pair programming）、伙伴复查（buddy review）、同行复查（peer review）、走查（walk-through）、正式代码检查（formal code inspection）都可以。复查能够为项目方方面面带来好处。项目经理要为团队提供多种方法。这些方法以特定顺序介绍：从最不正式到最正式。以我的经验来看，也可以从最有效和易于保持的，到最没有效果和难以维系的。

结对编程。先不要假定“这里没人愿意这么做”，要允许大家这么做。（而且可以提醒大家，他们已经完成过结对调试了。）我会问大家谁自愿进行结对。我建议他们在开始之前，先阅读一些相关资源，比如［WK02］和［SH06b］，还有http://www.pairprogramming.com。

不出意外的话，会有人愿意尝试结对的。相对独自工作来说，他们可以学到如何通过结对编程使工作变得更有效率。

除了减少缺陷、加快开发速度外，结对编程还能带来很大的好处：有两个人完全熟悉并知道如何处理同一块代码。而且，如果人们经常切换不同的结对对象，他们就会深入了解目前团队正在开发的系统的各个部分。此外，对代码作者的反馈也没有延迟。

用哪种生命周期都没有关系。结对这种技术总是可以用来让更多人了解代码。

伙伴复查。伙伴复查无法达到与结对编程同样的学习效果。没错，每个人都可以了解产品某个功能的代码，但是不会像作者那样深入。作者得到的反馈也会有一点点延迟——即完成复查所需的时间。

同行复查。同行复查与伙伴复查的意思差不多（把你的代码给别人看看），但是很多人倾向于一次复查一个完整的模块，不管是一个文件还是几个文件。复查大量代码更难，很难找出需要进行复查的大块时间，而且很难记住脑子里面所有的想法。

同行复查很难达到与伙伴复查同样的学习效果。以我的经验来看，这样做经常只能复查代码风格，而不是内容。对于作者的反馈延迟可以长达一星期。

走查。用走查的方式，一些人会集中在一个房间内，由代码作者说明产品，过一遍文档。这里几乎没有团队学习的过程，而作者得到反馈的延迟时间也相当长——也就是用来组织会议需要

的时间。

正式检查。如果做得好，正式检查可以帮助团队通过讨论了解产品。不过我很少看到正式检查能够在组织内长久实施下去。即使是启动检查的人，也很难维持检查的动力。

维持检查很困难，因为检查别人的代码会打乱每个人的节奏，还有项目的节奏。要想进行一次费根式（Fagan-style）的检查[1]，人们必须离开自己的工作任务上下文，详细阅读产品代码，准备评论自己看到的东西。我发现，为了一次两小时的检查会议，要花上几小时甚至一天的时间去准备。

9.5 准备重构

重构是对代码的简化，无论是产品代码还是测试代码。重构不等于重新设计，只是简化而已。重构过的代码不会改变它的契约（contract），只是更简单。

我的代码不见了

——哈尔，初级开发人员

我现在参与的项目是离开学校后的第二个。在第一个项目中，经理听了我的代码工作估算后说：“好，既然你还是个新人，咱们就再多加点儿时间吧，这样你可以把积累的经验用到写好的代码中。”我觉得他这么做纯属多余，不过没关系，我还是加倍努力，在截止日期来临之时完成了手上的工作。接下来，我了解了整个系统真正的运转方式，就得修改已完成的工作，不只耗费了所有额外的时间，而且还多用了一点点。让我感到惊讶的是：为了解决问题，我没有编写更多代码，而是简化了现有做法并去掉一些代码。我的代码不见了。

现在这个项目，我使用了持续集成，而且一边开发一边重构。在开发时，简化并清洁代码，而不是改变设计，这让我对自己手上的工作和开发速度有了更清晰的认识。而且，我的代码仍然在继续消失。

如果你曾经随项目进展计算过代码行数，要是项目采取顺序式生命周期，或是将集成和测试放在项目临近结束时进行，那就会看到一个S形曲线（见图9-1）。可以注意到，开始集成和测试后，代码的数量就开始减少了，这是因为发生了重构。（没错，也许是重新设计了某些东西，不过以我的经验，主要是重构造成的。）如果不准备在项目快结束时进行重构，或是不愿意在顺序式生命周期中偿还技术债务，代码量会居高不下。在项目快结束时，代码缩减就不会发生。

[1] 费根式检查（Fagan inspection）是一种在开发文档中试图发现缺陷的结构化过程。这些文档可以包括代码、需求说明、设计文档以及其他软件开发过程涉及的文档。其命名来源于迈克尔·费根（Michael Fagan）——正式软件检查的发明人。详情可查看：http://en.wikipedia.org/wiki/Fagan_inspection。——译者注

9

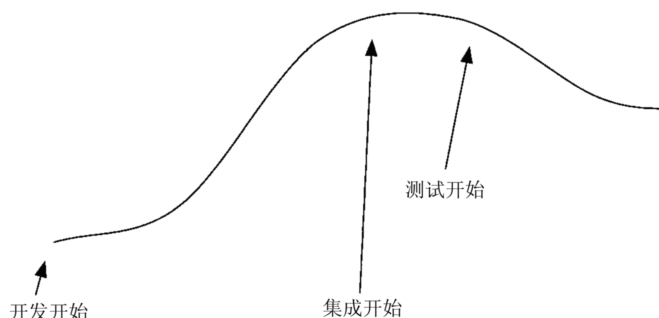


图9-1　顺序式生命周期中代码量的典型变化

在使用敏捷生命周期的项目中，代码量的曲线很可能如图9-2所示。代码的增长要慢得多，因为开发人员仅仅编写当前功能需要的代码。他们还会一边开发一边重构，而不是等到项目快结束时再去修复一大堆bug。（对于很多bug来说，去掉某些代码就可以解决。）

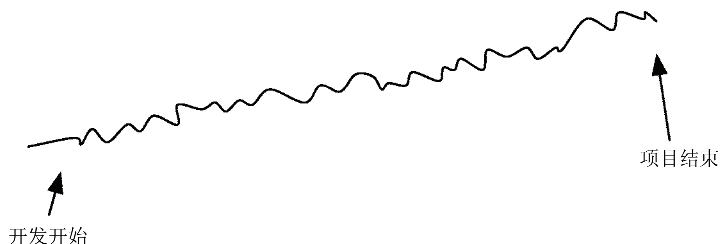


图9-2　敏捷生命周期中代码量的典型变化

项目经理可以让重构与开发同时进行，也可以最后再进行重构。如果希望发布的产品中缺陷越少越好，就必须对代码进行重构。如果让重构与开发同时进行，重构的成本是非常小的。如果打算在项目结束时重构，那成本可就高了，而且你会发现总是没有时间去做重构。要么管理层就会指示你不要重构。高昂的成本来源于开发人员得到的反馈延迟过久，以及不知道应该重构哪些代码以及如何重构，因为开发人员很久没有思考过要重构的代码了。

9.6　通过用例、用户故事、角色和场景来定义需求

要减少不必要的代码，有一个好方法：想想谁会使用系统，又该如何开发用户需要的系统。

有很多项目都试图通过定义功能性和非功能性需求来确定需求。可是这些需求没有说明一个人如何使用系统，以及一个功能在何种场景下必须运行。需求收集的方法应该为项目团队提供上下文，这样才能深入理解需求。

谁需要那个支票账户？

——克拉丽莎，资深经理

我的项目团队陷入困境。他们实现了一堆只开发完一部分的功能，可是没有哪个能够真正运行。我召开了一个会议，以决定接下来该做什么。

在会议上，我想知道：为了完成项目，哪些用户是他们的首要服务对象。他们看着我，脸上毫无表情。“你们看，我们是一家银行。我们希望年满18岁、将要进入大学的人们首先使用我们的服务。还有住在郊外的妈妈们，她们还有其他资产，75岁的老奶奶，以及年方15已经靠做家务挣得零花钱的孩子们。我们可以为他们提供不同的理财服务。如果他们走进来，我们希望他们成为我们的客户。对于这些不同类型的客户，我们真地想过要给他们提供什么样的产品吗？”

他们以前太过专注于系统的内部功能，忽视了系统真正要为之服务的用户。我们把期望捕获的用户重新进行了排序，项目团队因此得以完成需求定义，并完成系统的功能。

人，总是人的因素，不是吗？

只要大家了解需求的上下文，开发人员、测试人员和文案人员就能知道如何开发、测试、编写文案。如果他们不能了解，要是需求以功能性和非功能性需求的方式罗列，他们也可以问一些更好的问题，从而深入了解需求。

9.7　分离需求与 GUI 设计

我们希望系统解决的问题通过需求得以体现。GUI设计是要体现出GUI如何引导用户使用系统以解决他们的问题。很多项目都以需求获取为幌子，陷入GUI设计的瘫痪之中。这很让人惊讶。如果你的项目总是陷于无尽的需求工作之中，看看问题是不是出在GUI设计上。

GUI是设计，不是需求

——凯伦，程序经理

我在一家新公司工作时，试图拯救一个陷入“需求地狱”的项目。需求文档已经达到300多页，而且远未完成。

阅读这些文档后，我找到了原因。所有的GUI设计都被记录在需求文档中。GUI设计没有放在项目的设计阶段，业务分析人员和GUI设计人员试图将所有的GUI需求都放在需求文档中。他们使用了功能强大的图形设计工具，并在需求文档中定义GUI。

我向他们询问原因，他们看着我，说道：“这些是GUI需求。”我建议他们认真看看GUI设计，并且考虑这些设计是否应该跟希望系统解决的问题放在一起。GUI设计不应放在需求文

档中。

最终，他们同意采纳我的建议，我们也可以逃离需求地狱了。而且，由于我按照逐个功能重新组织了项目，GUI设计也就跟各个功能结合在一起了。我们定期检查整个GUI的一致性，但是这与需求无关，这属于设计。

人们很容易在项目开始阶段设计GUI，并称其为需求。如果要这么做，项目就永远无法找到自己的节奏。它会一直陷于需求的泥沼之中，直到最后，无法完成任何客户需要的功能，虽然到时候能够得到精美无比的GUI。

9.8　尽可能使用低保真度的原型

使用低保真度的原型，人们可以更全面地评估要解决的问题。高保真度的原型会限制反馈[①]。在项目初期，项目经理希望得到更广泛的反馈，而不是要限制反馈。用纸制作原型与使用即时贴安排日程类似：它们可以保证团队全心投入。相对于电子文档来说，人们也更愿意在纸面上考虑更多的选择[②]。

纸上的原型可以节省时间

——凯伦，程序经理

当我们走出“需求地狱”后，我意识到UI设计人员开始拖项目的后腿。经过调查，我发现他们的原型不是在纸上完成的，他们使用带有新logo的漂亮图片，包括所有的图形内容。他们没有首先设计工作流程，而是直接开始设计图片。

我向他们说明，纸上的原型可以帮助人们看到流程并做出评价。要添加logo和其他的图形，时间上不成问题，但是工作流程会影响架构。如果工作流程设计正确，到了项目尾声，GUI方面的变更就会少很多。

UI设计人员对此怀疑，不过还是答应尝试先在纸上完成原型，然后再使用线框图，最后用图形工具把漂亮的图形放进去。到了项目快结束时，在工作流程上的变更就少多了，因为每个人都能理解项目本应具备的工作方式。他们也会对实际的图形做调整，但这些变化不会影响到最终的产品。我们也做到了按时发布。

低保真度的原型不只可以用在GUI设计中。我参与过这样的团队，为了解决难搞的定时问题，他们就几种不同的架构设计争论不休。我们用了一个很大的会议室，大家排成几队，对应于我们要考虑的几种队列，还用一些纸记录了我们要变换和移动的数据。项目经理开始模拟，大声叫出

---

① 请参见http://headrush.typepad.com/creating_passionate_users/2006/12/dont_make_the_d.html。

② 请参见http://www.uie.com/articles/prototyping_tips。

事件的顺序——说明我们应该何时并向何处转换或移动数据。通过这样的方式，我们发现了一些从未想到的时间和资源竞争的问题。完成了人的模拟之后，我们也开发出了程序原型。不过我们更深入理解了风险所在。

铭记在心

- 项目经理可以邀请团队成员考虑这些实践，不过你不能强迫他们采纳这些实践。
- 如果项目经理即使尽全力也只能选择一个实践，推荐选择持续集成。
- 项目经理采纳和调整的实践要有助于保持项目的节奏，允许项目提高启动和结束的速度。

第 10 章

管理会议

作为项目经理，你很有可能会在会议中消耗一整天的时间。只要参加或是召开的会议的确有其价值，这就没问题。然而，项目经理并不能决定哪些会议是有价值的。在本章，项目经理可以学到如何分辨会议的好坏，以及如何将更多的时间用在有效率的会议中，如何尽量不参加毫无价值的会议。

有许许多多项目经理在会议中浪费时间。你的时间、项目团队的时间，它们太宝贵了，根本不应浪费在会议上。不要在会议之间穿梭，要判断哪些会议值得你花时间。你必须知道要召开哪些会议、参加哪些会议、派代表替你参加哪些会议，以及——最重要的——忽略哪些会议。

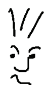
提示：发现并摧毁浪费时间的会议

项目经理在组织项目时，要注意浪费时间的会议。如果找到了，取消之。要让团队将注意力集中在项目上，这是最简单、最有效的方式 [DeM01]。

10.1 取消这些会议

在帮助团队朝着合理的交付截止日期前进时，保护团队不受外界干扰和影响，是项目经理的职责所在。项目经理可以帮助团队不参加不必要的会议。如果会议对于任何人都毫无价值，那就取消掉；同时允许团队成员不参与无法贡献和收获价值的会议。也许有些团队成员会不高兴，因为他们认为你觉得他们不够重要从而不能参加会议。要跟他们强调，你不让他们参会是因为他们太重要了。

项目经理要避免让团队成员参加毫无效率的会议，同时也要记得自己不去参加这些会议。总而言之，如果你在参会过程中根本没有听别人在讲什么，自己也没有做出贡献，那你就对不起付给你的那份工钱。



小乔爱问……

这个会议有必要吗？

下次开会的时候，你可以看看周围的人，计算一下这个会议的成本。假如有12个人参加，其中一半年薪80 000美元。嗯，年薪80 000相当于每小时40美元。这6个人参会一个小时，成本相当于240美元。这还没有计算他们会后再投入各自工作以及会前准备所占用的时间。

把其余参会人员的成本也加起来，别忘了算上你自己的时间所耗费的成本。这个会议为公司提供的价值最少能到这个数吗？如果与会者不参加这次会议而是去工作，这样所创造的价值，也就是丢失的机会成本，我们甚至还没有计算在内呢。

10.1.1　避免不需要你解决问题的会议

很多组织每周都会有成千上万“状态报告”之类的会议。如果不需要你参与决策或是解决问题，那这样的会议就不必参加。真的，我不骗你。

可能有些人会想：“呃，JR。我所在的公司可不能这么做。要是你不在会议上露面，就等于你做的事情无关紧要了。”可能你所在的公司是这样，项目经理能够提升，是因为他们在公司中表现得显眼，包括在开会的时候。要是这样，你就得想想是不是要改变这种文化，是要继续在会议中浪费时间，还是转身离开。请看7.7节。

不过一般来说，以前要开会通常都是有缘由的（数年之前），可现在这个缘由已经不成立了，剩下的就只有会议。你可以鼓足勇气，表明自己要与项目为伴，所以你很希望能看到会议记录，但是不必亲自到场。

尝试一次，看看效果如何。如果还不错，那你就可以继续选择自己要参加哪些会议。要是不可行，那就等过一段时间再尝试。身为项目经理，你要评估并引领项目取得成功，而不是任由会议妨碍你的工作。

10.1.2　绝对不要举行多人参加的顺序式进度报告会议

绝对不要举行顺序式进度报告会议。绝不。

项目经理坐在那里，聆听每个人报告自己上周做了什么、接下来这一周要做什么，这就是顺序式进度报告会议。除了发言的人和项目经理之外，每个人都觉得很无聊。顺序式进度报告会议等于浪费团队成员的时间。这会鼓励大家“心有旁骛”：阅读邮件、给别人发即时消息、浏览网

页……所有这些对于推进项目工作毫无裨益。团队成员不会注意别人在说什么，他们也不会专心参与这个会议。顺序式进度报告会议根本不是会议，更像是一种注重形式的仪式［DL99］。

如果项目经理主持的项目团队会议不能解决任何问题，那这个会议就是在拖项目的后腿。赶紧停止吧，重新规划相关会议。

“可是，JR，我的团队成员非常希望知道别人在干什么。”以我的经验，在小项目中，人们都已经知道别人在做什么了。项目规模比较大的话，没人关心这些细枝末节。别人的工作能否满足自己的需要，才是他们真正关心的。

如果项目经理相信自己的团队成员喜欢进度报告会议，那就首先主持一个不需要报告进度的会议。然后使用ROTI［RD05］或是以匿名的方式询问别人，看看他们喜欢什么样的会议。如果他们还是对进度报告会议情有独钟，那就开吧。要留心持续时间，并确保这些会议一直能够对团队物有所值。

10.1.3　避免这些会议

下列会议是要避免的。

没有存在理由的会议。有些会议似乎从远古时代就已经存在了。会议最早的负责人早已离开了原来的职位，最初的召开意图也已经消失不见。剩下的只有会议。不要参加这些会议。一般来说，人们都不会注意到你没有到场。

没有会后行动的会议。你的一位同事马上要召开一次会议。大家坐下来讨论问题。你甚至对某些事情有决策权。可是没有会后的行动计划。项目经理也不必参加这类会议。别人又怎么可能知道你没有参加呢？

上层主管的顺序式进度报告会议。好吧，这对项目经理来说可能有点难度。你的主管仍在举行顺序式进度报告会议。由于项目经理负责管理独立的项目，根本不必关心张三李四在干嘛。

略过这些会议，用电子邮件告知别人你的工作进度。还要记得告诉主管你对于这类会议价值的看法。

任何不允许携带笔记本电脑的会议。艾伦介绍了她所在公司的一个会议：“这些会议实在是太无聊了。我本来不应该带着我的笔记本电脑或是PDA。我本该坐在那里，专心聆听，而不是无聊透顶。”

跟负责会议的人单独碰面，说明你需要用笔记本解决问题。告诉对方你的想法，解释说你必须拥有必备的工具，才能提供最有成效的帮助。如果对方告诉你他希望聆听每个人的工作进展，

那就告诉他还有其他可以获取工作进度的方式（见10.5节）。

10.2　举行下列会议

无论你是什么类型的项目经理，为了项目团队考虑，下面这几种会议是应该举行的，而且可以跟项目团队一起开。

- 项目启动会议
- 发布版本规划会议
- 向高层报告进度的会议
- 项目团队会议
- 迭代回顾会议
- 项目回顾会议。请参见15.4.3节。

10.3　项目启动会议

项目启动会议是项目经理的第一个项目团队会议。稍加用心，项目经理就可以为整个项目设定积极的基调。

如果项目章程还未确定，项目经理可以利用项目启动会议跟团队一起编写项目章程。项目经理了解项目的驱动因素（参见1.4节），因此可以跟大家一起确定项目章程（参见1.6节）。

如果章程已经编写完毕，那就跟大家一起把章程过一遍吧。这可以帮助大家看到如何开始复查工作成果，请参见9.4节。

10.4　发布版本规划会议

如果项目经理管理敏捷项目，会主持一个发布版本规划会议，而不是项目启动会议。发布版本规划可以让每个人（包括团队成员、出资人、客户）看到你对于项目进展的规划。项目经理会规划出希望一个迭代交付哪些功能。项目经理会在迭代之间对产品待办事项列表重新排序（参见16.6.2节），细节会有变化。在项目初期，这是你所能做出的最佳预测了。

首先，项目团队会利用规划扑克（参见5.1.9节）来估算功能的相对大小。团队接下来会估算开发速度，并预测需要多少个迭代，以及每个迭代中完成哪些工作。要特别注意的是，如果团队刚刚接触迭代式开发，对于每个迭代都要交付可供发布的软件这一点，他们也是刚有一点概念，

他们对自己每个迭代所能完成工作的初步估算会不太准确。这没关系，项目经理可以随着迭代的展开测量团队的开发速度。到了一个迭代结束时，你会知道是否需要马上再进行一次发布版本规划，或是等过几个迭代后再重新规划。

团队估算完功能的相对大小后，客户或是产品负责人就要对功能进行排序了。团队会把这些功能放到迭代中。可能要过几个迭代之后，团队和客户才能最终确定哪个迭代开发哪些功能。项目经理要帮助团队和客户把注意力放在最初几个迭代上，而且你也要做好重新规划的准备。

在发布版本规划会议结束时，项目经理应该得到下列成果：对于完成的定义、可能的发布日期、可能已经按迭代组织好［Coh06］并且有大小的待开发功能列表。

10.5　进度报告会议

进度报告会议分两种：项目经理和团队成员之间的一对一会议；项目经理和上级主管之间的会议。你可能注意到了，我在前面的列表里没有提到团队进度报告会议。原因在于：如果不是每日站立会议，多人参加的进度报告会议就是顺序式进度报告会议，纯属浪费时间（见本章开头的提示：发现并摧毁浪费时间的会议）。

要从团队成员那里获得项目进度，我推荐使用下列技巧。

- 每日站立会议（参见10.5.1节）。
- 每周的一对一会议，在项目经理和每个团队成员之间进行（参见10.5.2节）。
- 每周的电子邮件进度报告（可与10.5.4节比较）。

10.5.1　每日站立会议

如果项目经理使用的是敏捷生命周期，可能会使用每日站立会议，以了解项目进度。实际上，只要团队成员创建的任务足够小，任何项目团队都可以使用每日站立会议。如果项目经理没有使用“小石子”或是其他方式将任务拆分成1~2天可以完成的小部分，站立会议就是没有意义的。这时，你最好通过一对一方式，帮助团队成员分解任务。任务持续时间越长，项目经理就越有可能遇到“90%完成状态”日程安排游戏（参见6.14节）。

下面是每日站立会议的运作方式。团队中所有人在一个特定的时间都聚在同一块区域，耗时不超过15分钟。每个人都站着。（如果在会议室中召开站立会议，人们会坐下来，那么会议就将持续1小时。）轮到每个人的时候，他要回答下面三个问题：

- 我昨天完成了什么？

- 我今天计划要做什么？
- 我面临哪些障碍？

项目经理会发现这些会议用不了多少时间。因为讨论的是一天的工作量，所以在15分钟之内完成并不是问题。如果很难空出15分钟时间，请参见10.9节[①]。

有些人会说："好吧，理论上看起来不错。但是我的团队有24个人，我该拿他们怎么办？"如果你管理着团队领导或子项目经理，而他们也管理着人数更多的项目团队，那他们就应该跟自己的团队成员开站立会议，并将障碍报告给你。

"啊，可是这24个人都直接向我报告啊。"很抱歉，我要说的会让你失望了，你无法仅凭一人之力直接管理24个人。你的团队会自行拆分成更小的小组（这没关系）。问问你自己这些问题：让这些人直接跟我报告，我能得到什么？我能通过其他方式获取项目进度，并帮助团队更好地工作吗？大多数情况下，如果项目经理手下没有技术带头人，他们自己就会成为项目的瓶颈，因为他们担负了太多决策的责任。

在一个项目团队中，我建议直接向项目经理报告的人不超过6个（参见7.4节）。如果参与项目的人超过8个，他们趋向于自组织成更小的团队，也许是按职能，也许是跨职能的。让团队自己组织吧，项目经理可以因此而收获更加顺畅的沟通。

"可是我该如何跟每个人建立联系呢？"除非项目的迭代周期很短（一两周），跟每个团队成员进行每周一次或两周一次的一对一会议是值得的。另外，团队的带头人们每天会互相沟通，彼此提供反馈和小幅度的调整。

10.5.2　一对一会议

如果项目经理管理的项目使用顺序式或迭代式生命周期，或者处于增量式生命周期的早期，你应该与团队每名成员进行每周一次的一对一会议。换句话说，要想尽早对项目做出预测，项目经理就得尽早动手跟团队成员做一对一的沟通，验证项目的状态。（如果你在管理大型的项目或是工程，那每个项目带头人就要举行这些会议，而你要跟这些带头人单独碰面。）在敏捷生命周期中，项目经理不必跟每个人碰头，上级主管要承担这个责任。如果你还担任职能经理，请查阅［RD05］以了解其他形式的一对一会议。

如果项目经理管理矩阵式团队，要小心你在一对一会议中讨论的议题。在一对一会议中只能讨论项目的进度，而不要涉及职业发展。做好准备，为对方在项目中的工作提供反馈和指导。要

① 也可以查看杰森・叶的文章：http://www.martinfowler.com/articles/itsNotJustStandingUp.html。

小心你的一对一会议不要跟职能经理的一对一会议重叠。先跟职能经理沟通。为团队的每个成员设定目标和工作边界。定期与职能经理会面，确保你们仍然可以互相协调彼此的工作。

下面这些一对一会议的形式，即使你不是人力资源经理，仍然可以使用。

- 问候。当一个人走入你的办公室，你要说“你好”。通过问候，可以将气氛调整到一对一的情境下。还要记得关闭手机，离开电脑。
- 讨论进度和状态。此时可以复查对方的“小石子”任务完成状况。如果项目经理一直在试图告诉对方如何制订“小石子”任务，此时就是指导的最佳时机。如果项目采取的生命周期贴近顺序式，那项目经理就应该尽量用可见的形式把项目进度表示出来。人们很容易陷入某种困境却不自知。
- 讨论他们的障碍。项目经理应该帮助人们认识到他们面临的障碍。我见过这样一个项目团队，他们已经习惯于花上几个小时来设定项目环境，并没有认识到这其实是一个障碍。
- 复查所有的行动计划，包括你的和对方的。如果项目经理在之前的一对一会议中有一些亟待实施的行动计划，要报告自己的进度。在一对一会议中，要做笔记，记录会后的行动计划和眼前发生的问题。

一对一会议的目的是要让项目经理看到项目的进度，同时让团队成员告诉你他们需要帮助。如果项目经理仅仅公布项目进度，人们会倾向于自己完成任务、解决问题，即使他们已经陷入困境。项目会逐步走向“日程小鸡”游戏（参见6.13节）。私人的、一对一进度会议能让项目经理知道人们遇到问题了。要想了解人们是否遇到问题，最好的方式就是明确了解他们的工作进度。

10.5.3　通过可见的方式了解进度

让人们说明他们的工作进度，包括他们下周要做的工作以及他们如何跟踪进度。要求大家将自己的任务分解成“小石子”级别的待办事项列表。还要说明，你不会把他们的“小石子”放到项目日程安排中。“小石子”会让日程安排变得复杂，而且项目经理也不能像保姆一样去详细了解每个人的工作状况。要说清楚，你的工作是要了解大家的工作何时取得进展、何时陷入僵局。“小石子”能够让人们监控自己的工作状态，同时让项目经理有所了解。

让人们注意自己何时陷入困境，然后告诉项目经理他们是否需要某些帮助。请求帮助并不丢人，独自挣扎才是不合适的。如果有人在进行一项大产出的工作，让他们想想能给你展示什么：做好标记的临时设计、算法的性能度量数据、丢弃的脚本数量，还是其他能够展示进度的东西。由于团队成员自己负责自己的任务和可交付物，所以当他们需要帮助时，你所做的就不在微管理

（micromanage）的范畴了。

项目经理偶尔会遇到这样的团队成员，比如戴夫，他认为自己完成工作不需要别人干涉。要是发生类似情况，告诉戴夫：如果他坚持自己这方面的隐私，那你就不知道如何有效管理自己的项目了。问问他愿意展示给你哪些信息，以表明他的进度。这不总有效，还不如直接询问他的工作何时可以完成。

如果戴夫告诉你一个超过两周的日期，要跟他说这太久了。在很多项目中，如果有人用多于一周的时间完成某项工作，项目经理还可以承受。可要是有人用多于两周的时间才能交付某项工作，而整个项目还不会因此而受影响，这样的项目我可从来没有见过。如果你相信戴夫没有恶意，可以问问他是否能在少于两周的时间内交付工作。也许你可以等到两周过后看看情况如何。（如果有人每两周成功交付一些工作，我仍会感到不安，不过还是可以接受的。）一旦戴夫错过了截止日期，跟他谈判，商讨一种跟踪任务和进度的不同方式吧。项目经理必须知道人们在不断取得工作进展。

如果项目经理能够说明为什么需要信息，以及需要什么级别的信息，那么大多数团队成员都会愿意跟你合作。你会以可见的方式了解项目状态。

10.5.4　通过电子邮件，从团队成员那里获取每周进度报告

如果使用敏捷生命周期，项目经理就不必从团队成员那里获得每周的进度报告。你已经从每日站立会议中获得足够的信息了。由于这些信息足以产生项目的仪表板，你也就不再需要更多信息了。

对于其他生命周期，我会通过电子邮件的方式，要求团队提供每日站立会议中同样的信息。而且，我会让他们提供后续几周的“小石子”任务。没错，这是在让大家做波浪式规划。项目经理会发现这样做是有效果的。强迫大家将大任务分解成小的可交付任务，并预计未来几周的工作，他们会（你也是）了解项目是在迈向成功，还是灾难。

下面是我的电子邮件进度报告模板。

工作成果。几段简短的话（每段两三句话），报告过去一周完成的工作。

未来的里程碑。任务描述　　规划完成日期　　预计完成日期　　实际完成日期

有些项目由于发生外部事件，团队成员的工作会有变化。对于这样的项目，可以在右边加上另外一列：估算变更次数。

障碍。团队成员写下导致不能完成工作的障碍。而且希望能从这个列表产生行动计划。

10.5.5　每周向团队报告进度

在顺序式、迭代式和增量式项目中，可以每周发送一封电子邮件，告诉大家项目的状态。把每个人报告给项目经理的进度汇总起来，不要提到他们所说的障碍。这样一来，项目的透明度就提高了，也不必召开顺序式进度报告会议。说明项目仪表板（参见第11章）的位置以及其中数字的含义，让大家专心于最终的目标，同时还要对中途的里程碑有所注意。

在敏捷生命周期中，项目中每个人都对进度了如指掌。每天的站立会议中，大家会说明自己完成的工作、面临哪些问题以及自己将要做什么，项目经理不必发送每周的电子邮件。如果必须要为管理层准备进度报告，也要把这封电子邮件发送给项目团队。

10.6　向管理层报告进度

要让出资人参与到项目中，定期给他们发送进度报告是一种不错的方式。即使你的主管习惯采取微管理的方式，定期为他们提供数据也许可以阻止他们管得过死。

如果使用有时间限制的迭代管理项目，要是项目经理不能说服管理层参加迭代结束时的演示，你只需在迭代结束时给他们发送进度报告。每个人都能在迭代结束时亲眼看到产品，也就没有必要像其他生命周期那样准备进度报告了。针对其他的生命周期，只需每周或每两周发送一次进度报告即可。管理层要是不习惯有时间限制的迭代，那就针对三四周的工作发送中期进度报告。

进度报告可以从类似天气预报的项目仪表板的概述开始。包括已完成工作成果的列表，尤其要标示出完成了哪些中间阶段的里程碑。请他们去看项目的仪表板，并欢迎他们随时提出疑问。

如果你的主管喜欢看到所有的详细数据，要确保从项目仪表板中收集所有的详细数据并发送给他。也许你要迁就一下这位主管，定期召开会议，跟他讨论数据。别以为可以躲得过主管对于数据的要求，你做不到。最佳的做法就是整理数据，并准备跟主管一起讨论。

10.7　项目团队会议

要将项目团队会议作为解决问题的会议。如果有要讨论的会议提纲，应确保这些提纲是为了解决某个问题，或是阻止某个问题的发生。

下面这个提纲模板的目的就是帮助人们把注意力放在解决问题上，而不是为了报告进度。

项目团队会议提纲

标题行。（项目名称）团队会议提纲、日期、时间、地点。

期望与会人员。列出他们的姓名。

主要里程碑检查。对于任何非敏捷的生命周期，列出主要的里程碑以及期望何时达成这些里程碑，这有助于人们在开展自己工作的同时了解上下文。如果希望了解估算质量因子EQF（见11.2.4节），此时最适合。要加上你对于讨论这项议程的预计时间。

本周遇到的问题。如果项目经理有特定的问题需要团队解决，可以列在这里。要加上你对于讨论每个问题的预计时间。

现有障碍？询问对于设备方面的要求，以及其他自一对一会议后涌现出的障碍，包含讨论该议程的预计时间。

其他话题？询问大家是否有想讨论的其他话题，包含讨论该议程的预计时间。

回顾以往行动计划。使用行动计划的简单列表：截止日期、负责人、行动描述，包含讨论该议程的预计时间。

下次会议。说明日期、时间、地点。

未决事项/停车位。列出项目经理和团队现在不想忘记而又不急于解决的事情，要按照准备处理每项的预计日期进行组织。

这个提纲只包含了需要整个团队讨论的条目。如果项目经理认为有些属于整个团队的问题但可以由少数几个人解决，要事先感谢他们以后的协助处理，再讨论下一个议题。

项目经理总是可以早点结束这个会议。很有可能值得讨论的议题费时不会超过一小时，这样很好。准时开始，早点结束。人们会感谢你的，他们会觉得这一天又额外多出了一个小时。

会后不要忘记给每个人发送会后行动计划。当人们完成了自己负责的行动后，他们应回复你的邮件，并说明什么时候做了什么。

10.8　迭代审查会议

迭代尾声，团队会向产品负责人展示工作成果。没错，这是一次演示。在敏捷生命周期的理念中，这是要告诉别人：我们要随工作推进不断检视并调整工作方式［Sch04］。这也是要向产品

负责人演示产品的原因。

此外，团队会讨论开发速度图表，还有其他在项目仪表板上的团队数据。最终，团队会进行一次中期回顾，从过去的迭代中总结经验教训。

如果项目经理使用敏捷方式，可以忽略其他会议，只关注发布版本规划会议、每日站立会议、迭代审查会议和回顾会议。

10.9　会议疑难问题解答

项目经理在开会时，可能会遇到下列问题，还包括如何解决这些问题。

每日站立会议超过15分钟。要保证大家都站着开会，一旦坐下来，会议就会拖得更久。

确保人们报告已经完成的工作。如果人们报告正在进行中的工作，他们要么是没有将需求分解成足够小的功能，要么就是没有将任务分解成“小石子”。

没人能够及时参会。如果这仅仅是你所在项目中发生的问题，就重新商量一个开会时间吧。也许选择现在这个时间开会，打断了人们的工作状态。我曾经选择在吃午饭的时候开项目团队会议（因为团队要求在午饭时开会）。我将时间定在12:15，同时说明不管他们是否到场，会议都将在12:15开始，而且在1:00结束会议。项目团队欣然接受，并且效果不错。

通常，这不是一个项目问题，而是组织层面的问题。我会在预计的时间开会，如果没有人到场，就自己一个人做出所有的决策，包括替那些要解决问题的人们做出决策。（没错，这看起来有点儿专横。）

项目经理要确保完成自己负责的工作，包括至少提前24小时向与会人员发送会议提纲、会后24小时内向与会人员发送带有会后行动计划的会议记录。

项目经理要确保不在会上浪费别人的时间。如果你在会上进行顺序式进度报告，而不是进行一对一式谈话，那你就是在浪费别人的时间。你可以询问大家，他们在会议上投入的时间［RD05］所获得的收益是否足以让会议继续进行。也许你召开会议过于频繁了，或者你没有解决项目的问题。

确保团队成员不会因为感到浪费时间而抵制你的会议。请参见本章开头的提示：发现并摧毁浪费时间的会议。

没人能在下次会议前完成各自负责的行动计划。确保在会后24小时内向大家发送会议记录。如果其中包含行动计划，要写明每项行动的负责人和截止日期。

对于下一次会议，至少提前24小时发送会议提纲给大家。这会提醒人们，他们还有待完成的行动计划。

与项目无关的人希望参加团队会议。明确说明项目团队会议的参与人员，是为了让人们知道谁会就项目相关议题做出决策。如果其他人希望参加，先寻找原因，然后为这个人设立一个角色。

如果有资深主管希望“帮助”团队更快完成工作，让他看看项目仪表板。确保这位主管了解团队的速度以及团队中成员的各自角色。如果团队在同时应对多个项目、处理多个任务，主管可以制订项目组合并设定优先级，以这样的方式帮助团队。

要是这位主管还是希望通过“激励”团队来“帮助”大家，可以告诉他：知识工作者的激励是来自内部的，而不是源于外部［Koh93］。如果主管还想说点什么，那就给他几分钟时间，让他在会上讲几句，然后请他离场。这样，团队就可以开始解决问题，加快项目进程了。

这些行动都不容易，而且需要项目经理的谈判、影响力，甚至是绥靖的技能。不过要记住，保护团队不受外界干扰是项目经理职责所在。为了保护团队，你应竭尽所能。

要是还有主管（或其他人）想旁听你的会议，为这些人设立一个“观察员”的角色吧。在会议室桌子旁边的椅子后面安排几把椅子，让“观察员”们坐在那里。告诉“观察员”，他们可以记录自己的所见所闻，但是不能干扰会议进程。我的提议是：“如果你们忍不住想说话，写张纸条给我。这是索引卡片，你们可以把问题或是意见写在卡片上给我。”如果“观察员”不能保持安静，作为会议负责人或是推动者，项目经理有权请对方离开会议室。

项目经理有时需要邀请其他人参加会议，好从他们那里获得对于决策的意见或是收集数据，使得项目团队可以做出更好的决策。此时，要在会前说明这些人的角色，以及你将如何采用他们的意见。

10.10　管理与远程团队的电话会议

项目经理有时无法召开与团队的面对面会议。如果需要召开电话会议，这一节的建议可以为你提供一些有价值的帮助。

电话会议的地狱现在变成了天堂

——文迪，一个全球项目的项目经理

我知道是怎么回事。我曾经有一个月经常主持电话会议，可是总不能开始通话，因为杰克在吃午餐，皮埃尔在进晚餐，人们总是很难在需要的时候聚齐，还有其他的原因。那真是个灾难。最终，我把规则定死了。

首先，我让主管购买高质量的扬声电话和头戴式耳机。你能相信吗？有一个人过去竟然用一种半双工的扬声器电话！真是不可思议！

然后，我提升了自己的引导讨论技术，同时制作更具力度的会议提纲。接下来，我告诉大家在会议中我们应该如何与对方沟通。每个地点都会有一个引导人，大家每周轮换担任该职位。我让一个管理员做笔记，这样每隔一段时间就可以复查我们的行动计划。

我开始强制推行“一次只允许一个人发言”和“吃东西时不许发出声音”这两项规则。大家都取笑我这种做法。不过还是起作用了。现在，我们的电话会议可以更顺畅地进行了。

下面这些提示有助于电话会议的顺利进行。有些提示可能很难制度化，要坚持，但不可鲁莽行事。

10.10.1 通用引导指南

如果项目经理引导的电话会议多于两个人，可以尝试下列指导方针。

- 作介绍，请各方介绍自己的名称和角色。
- 所有的参与者都应同意：一定的引导是必要的。
- 使用“一次只允许一个人发言”规则。如果有人在发言，允许对方讲完。如果在通话中出现了中断，要询问之前发言的人是否已经说完了。
- 项目经理在发言时要自我介绍。
- 如果提到某个人，要明确说出对方的姓名。
- 不要吃东西（特别是不要吃能发出声音的东西，或是包装会发出声音的东西）。如果只能在某人吃午餐或是晚餐时开会，要求对方在进食的时候将麦克风设成静音。
- 如果要翻译成其他语言，首先设定基本规则（在表述完每个完整的想法后再翻译），而且要强制推行。

10.10.2 准备工作

除了引导，项目经理还应确保参会的每个人检查了下列准备工作。

- 确保所用的电话上有“静音”按钮，而且这个按钮不会播放音乐，确实可以屏蔽你这边的声音。我曾参加过一次电话会议，听到有人说：“那是Johanna。她很强悍，不是吗？”我发声表示赞同，对方尴尬得不得了。
- 使用全双工的扬声电话。
- 如果你是电话会议的发起者，要保证在呼叫其他小组之前，你所在地点的每个人都在房间之内。

10.10.3　进行事先规划

举行电话会议，不妨先做一些事先规划。想清楚发生下列情况时该怎么做。

- 如果有人没能接入电话会议，而你也不知道他们在哪里，你该怎么办？我的一个客户有条规则：如果15分钟后人还是不齐，就重新安排时间。
- 如果联系不上其他团队，或是别人有事找你，该怎么办？除了电话会议的联系电话，告诉其他团队一个手机号码，让他们可以找到你。
- 人们应该知道你希望讨论的话题。提前发送提纲，让每个人都能看到，这甚至比面对面会议还要重要。在会议中，如果你需要修改提纲，确保大家都能知道你现在讲到提纲的哪个位置。
- 确保每个人都知道要讨论的话题。除了要提前选定话题之外，你还得保证话题不会过于分散。你还要先组织好与会者，确定他们适合讨论选定的话题。
- 公布会议目标，而且要保证目标清晰明确。
- 不断改变参与者发言的顺序（比如谁先发言，诸如此类），特别是对于要定期召开的电话会议。
- 如果没有特别好的原因值得开会，那就取消会议（要小心难以取消的会议）。
- 设定结束时间。坚守这个时间。
- 有必要的话，使用协作软件。

10.10.4　引导会议

要想在跨越地理位置和时区的情况下引导会议比较困难，可以考虑下列建议。

- 如果有人加入电话会议的时间比别人晚，让另外一个人（不是引导会议的人）总结目前的进展状况并说明谁在线上。这样，会议引导者就可以集中精力，只有两个人（迟到者和向他解释的人）会被分散注意力。
- 使用镜式反馈（重复发言者的话）。
- 使用集中式对话［RBS00］，这个技巧可以帮助人们分离客观数据、反应数据、各自的解读，以及各自的决策。
- 注意所有的讨论，这样你就可以知道讨论在什么时候陷入过多细节，知道何时中止特定的问题。
- 询问有没有人有不同意见（换句话说，可以问“是否有人不同意或不理解？”）。不要假定沉默就意味着默认。
- 要注意别人是不是从对话中走神了。

- 按话题组织提纲。一个话题讨论完毕，要跟大家确认。
- 要知道，想通过电话完全听懂非本国口音的发言是很困难的。
- 要确保人们的发言在电话会议中不会被打断。我也喜欢保留每个人的手机号，如果有人没有发言，我有其他方式可以联系到对方。
- 要在会议进行到一半和结束时进行总结。
- 在电话会议结束时，列出行动计划。
- 明确宣布会议的结束。

10.10.5　电话会议后的工作

虽然电话会议结束了，你还有下列工作要做。

- 一定要给所有的参会者发送会议中的决策、会后的行动计划，以此作为跟进。
- 确保自己知道电话会议何时真正结束。要检查扬声电话是否尚未关闭。

铭记在心

- 你可以判断一次会议对你是否有价值，是否值得花费你或者任何团队成员的时间。
- 要像逃避瘟疫一样，逃避顺序式进度报告会议。
- 观察会议进程，看看是不是能够满足每名参会者的需要。

第 11 章

创建并使用项目仪表板

11

项目经理面临的绝大多数问题最终都会归结到："我们的进度怎么样了？"问题可能来自于高层，他们想知道你能否在截止日期来临之时交付产品；也可能来自项目团队，他们想知道自己的进度如何。要不了多久，项目经理就会觉得：这就像是在长途旅行中，他们坐在汽车后座上，不停地问："我们到了吗？"

要想真正掌握一个项目，关键在于经常进行测量，包括定性和定量两种方式，并且要将结果公之于众。项目经理将测量结果作为项目进度的一部分展示出来，团队成员就可以调整自己的工作，并取得更多成果。

这些测量构成了*项目仪表板*。将测量结果放在一起，就能告诉大家团队的开发速度、整体进度、消耗和当前所在位置，这跟汽车仪表板的作用很像。

创建项目仪表板可以为团队提供反馈，同时让其他对项目感兴趣的人了解进度。可以使用"大的可见图表"（Big Visible Chart）或是"信息辐射器"（Information Radiator）[Coc04]，这样所有的人就都可以看到项目进度了。

11.1 测量有风险

测量项目会面临三个大问题：项目团队花费过多时间进行测量，从而影响正常工作；人们不认真对待测量；测量人，而不是项目。

很容易就可以发现，人们在测量上耗费了太多时间。你的项目团队成员是不是总在写文档、做测量，而不是进行与项目直接相关的工作？本章中的测量都是要由你——项目经理完成的。团队的绝大多数人不应该帮你获得测量结果。有些人负责管理配置管理系统或缺陷跟踪系统，你可能需要他们帮助。（如果项目经理需要测量性能或可靠性，就会需要开发人员或测试人员的协助。）要是你需要很多人配合，还是先加强项目在一些基础设施上的支持吧。仪表板的目标是为了使用数据来评估项目状态，而不是为了花费时间创建仪表板。

人们不认真对待测量活动，一般是很难发现的。不认真的原因通常是因为只测量了项目中的一种因素。项目经理可能会见到日程安排游戏（请参见第6章），或是其他无助于项目进展的行为。有一幅很著名的呆伯特漫画，其中老板承诺说他会付现金给修复bug的程序员。沃利（漫画中的一个角色）说道："我要通过写程序获得一辆小货车。"沃利打算搞上一大堆bug，然后再修复它们。如果只测量一种因素，就等于是在鼓励人们只注重这一件事情。要确保存在多种测量方式，以评估项目的真正进度。

在选择测量方法时，要确保以项目和产品为测量对象，而不是对准人［Aus96］。如果测量的任何东西可以直接跟踪到某个人身上，这就等于是在对人而不是项目或产品进行测量。测量人就等于是鼓励他们不认真对待测量，项目经理会因此难以准确地了解项目的状态，而且整个项目也有可能无法终结。绝不要测量人。

项目的有些方面很容易测量，比如项目启动日期、当前日期、预计发布日期，还有"我们已经完成了百分之*X*了"，因为项目团队就是用时间的百分比进行估算的。（请参见11.2.3节。）如果你只去测量日程安排完成情况，那就肯定不能在截止日期之前交付项目。

实际上，任何单一维度的测量都不足以全面反映项目状况。好比说你要开车去某地，也知道整个里程，可你却既不知道自己车的油耗，又搞不清楚还有多远的路程。在这种情况下，你不可能知道车上的油量能否让自己到达目的地。

要想看清项目真正的全貌，看看1.2节中提到的项目驱动因素、约束和浮动因素，根据这三者排列组合形成的6个维度，可从中选择出至少4个维度进行测量，再显示在项目仪表板上。这4个维度涵盖了项目经理最有可能修改的项目因素。如果不衡量它们，那项目经理就无从得知应该调整哪些因素，才能使项目取得成功。

提示：使用多维度的测量来评估项目进度

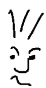
常言道："衡量什么，得到什么。"而且，正如11.1节所述，人们有可能不认真对待测量活动。既然你所测量的指标会产生最后的结果，就要保证测量足够多的项目信息，这样才能得到项目进度的真实反映。

工程副总罗伯给我打电话，他气急败坏。"JR，这些该死的测试人员！他们什么都做不好！"罗伯的项目有1 500个开发人员，还有大约350个测试人员。之前我已经见过一些测试人员，所以我说："有意思，不过好像我见过的人都自我感觉不错。""不可能，"罗伯语带讥讽，"开发人员每个里程碑都做到了，可测试人员一个都没有完成。我希望你赶紧做个评估。"

嗯，"开发人员能够完成每个里程碑"听起来有点可疑。我认识很多开发人员，即使是他们之中最棒的也不能做到总是按时完成。我以开明的态度开始评估，也许所有的1 500个程序员都

是无与伦比的。

不过我发现的情况是：开发人员只需向项目经理报告完成日期即可。这就是原因。项目经理只按日期考核开发人员，只按缺陷考核测试人员。除此之外，这个项目不测量任何数据。当我跟开发人员谈论他们的工作后，一切真相大白。开发人员丹尼表情痛苦，并解释道："我必须开始开发功能，因为甘特图上是这么安排的；否则，我的绩效评估就会打折扣。我只写好方法存根，这样就准时'完成'了。等测试人员报告一个问题，我就解决一个问题。"



小乔爱问……

我能重新开始已有的测量吗？

可以，不过我不建议这么做。因为导致问题的机制（流程和人）——比如项目晚于计划一个月启动——不会发现自身的缺陷。如果没有图表展示为什么项目从一开始就落后了，项目经理就无法改变原有的项目启动机制。

与其重新开始测量，不如在图表上画一条线，标明"原定启动日期"和"实际启动日期"，然后标明导致项目推迟启动的触发事件（见图11-7）。

罗伯所在组织的项目（和产品）完成得不好，无法按罗伯的要求交付。只要他坚持用单一维度来衡量团队的工作（只按日期考核开发人员，按缺陷考核测试人员），他们还会做出糟糕的项目。对罗伯来说，唯一的解决之道，就是让项目经理全方位测量项目，这样他们就能更准确说明项目的状况了。

11.2　根据项目完成度来衡量进度

驱动因素、约束和浮动因素这三者构成6种组合，项目经理可以从中选择几种衡量方式，根据项目的完成度来衡量团队进度。团队预先估算的准确性和团队已完成的进度，决定了项目完成度。不过仅从时间表上衡量还不够完善。要想准确判断软件项目的实际进度，应该衡量团队已经完成多少功能、已完成功能的质量以及还剩多少功能尚未完成。

11.2.1　使用速度图表跟踪日程安排进度

如果项目采取按功能逐个实现的方式，图11-1中所示的速度图表可以很好地指示出团队的项目进度[1]，而且它还能告诉项目经理剩余多少未完成工作。

① 请参见http://www.xprogramming.com/xpmag/jatRtsMetric.htm。

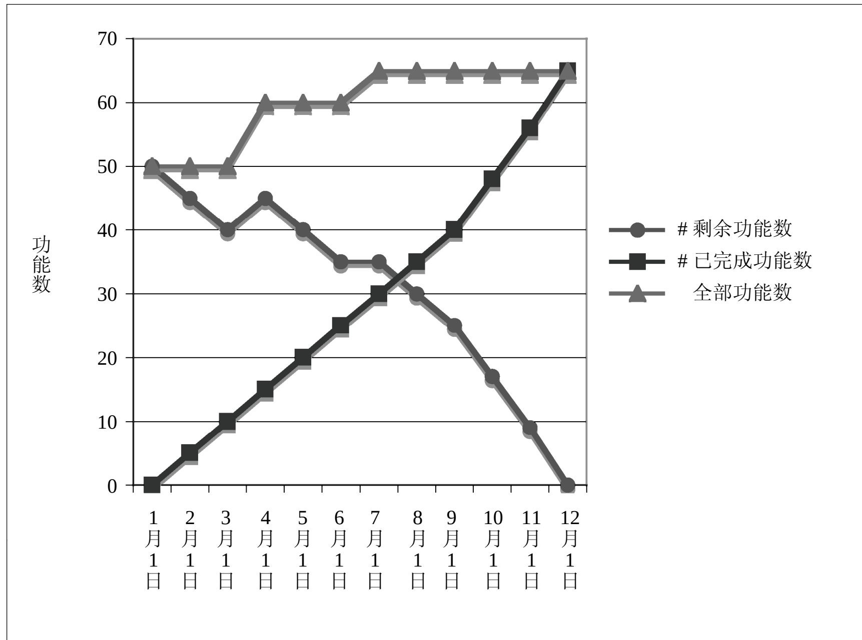


图11-1　项目速度图

可以按照下面的步骤来绘制速度图。将所有的功能汇总，得到项目的全部功能数。开发完一个功能，就将已完成功能数加1，并减少剩余功能数。如果必须在项目进行过程中添加新功能，就得把这些额外的功能加入到全部功能数中。即使这些功能的规模并不齐整，速度图也能有所帮助。

如果使用“小石子”切分任务，而且没有按功能逐个实现，那么跟踪“小石子”（请参见8.10节）的完成状况有助于项目经理了解目前进度。不过相比按功能逐个实现来说，这样做的准确性并不高。无论选择何种方式，不要仅询问团队成员是否达成他们的里程碑，还要记得检查已完成工作的质量。

提示：速度图是最佳图表

如果项目经理只能绘制一个图表，应该选择速度图。速度图集三种度量方式于一身（需求、已完成工作，还有时间）。虽然项目经理无法从中看到自己希望了解的是缺陷率或是成本，却能从该图中对项目的整体进度有所掌握。

这是因为你在一张图中同时度量了多个趋势：整体需求数量和已完成工作，其中包括所有的测试、文档以及项目需要的其他东西。这是最有用的图表。如果项目没有采取按功能逐个实现的方式，速度图中就不存在任何已完成的工作，而这正是项目的真实状态。速度图是项目经理的好朋友。

11.2.2　使用迭代内容图跟踪总体进度

除了使用速度图跟踪已完成功能之外，项目经理可能还需要详细了解每个迭代中的工作进展。（即使你没有使用带有时间盒限制的迭代，也不妨每过一段固定的时间就产生该图表。你会从中了解需求何时发生变化，缺陷何时出现。）

在图11-2中，可以看到发布的内容随时间变化的过程。在这个项目中，团队开始时的速度是每个迭代6个功能。进入到第9个迭代时，团队的速度降至2个功能，外加2个变更和4个缺陷。此时，项目经理意识到这样下去会出问题，于是停止在迭代中修改迭代的待办事项列表。团队的速度也因此在最后三个迭代有大幅提升。

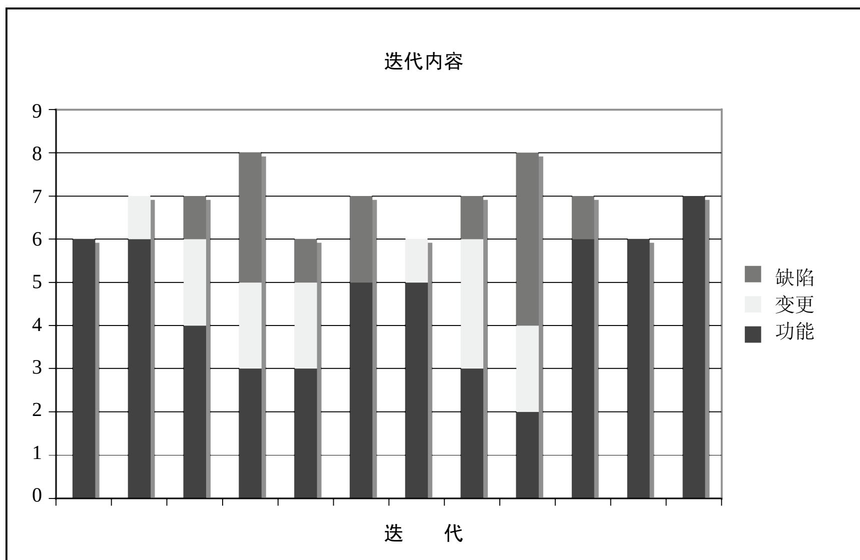


图11-2　一个项目的迭代内容图

在项目经理做出这张图之前，没有人了解迭代中发生变更的成本，也不知道这些变更会引入哪些缺陷。

11.2.3　计算软件项目的挣值并无实际意义

挣值可以用来衡量到目前为止已完成工作的价值。[①]然而软件总是在不断发生变化，要想计算出真正的挣值几乎是不可能的。如果无法明确定义一个指标，也就无法真正测量该指标。应该

① 来自© 2007 R. Max Wideman, http://www.maxwideman.com; 经许可后复制。

抵制组织要求项目经理报告挣值的企图。一个有形的产品，计算其挣值很简单。如果你要打造一个桌子，可以计算出物料的成本和花费的时间，看看桌子腿和桌面在拼合起来之前是否有价值。然而软件的挣值有所不同。

举一个例子。假设项目经理要在10个星期之内完成5个需求。设想你和项目团队认为整个团队在每个需求上要花费2周的时间。而且，假如团队由5个人构成。你的估算就是每个需求要花费相当于1个人10个星期的饱满工作量，总计需要1个人50个星期的饱满工作量。假定团队已经完成了前三个需求，包括测试，如图11-3所示。

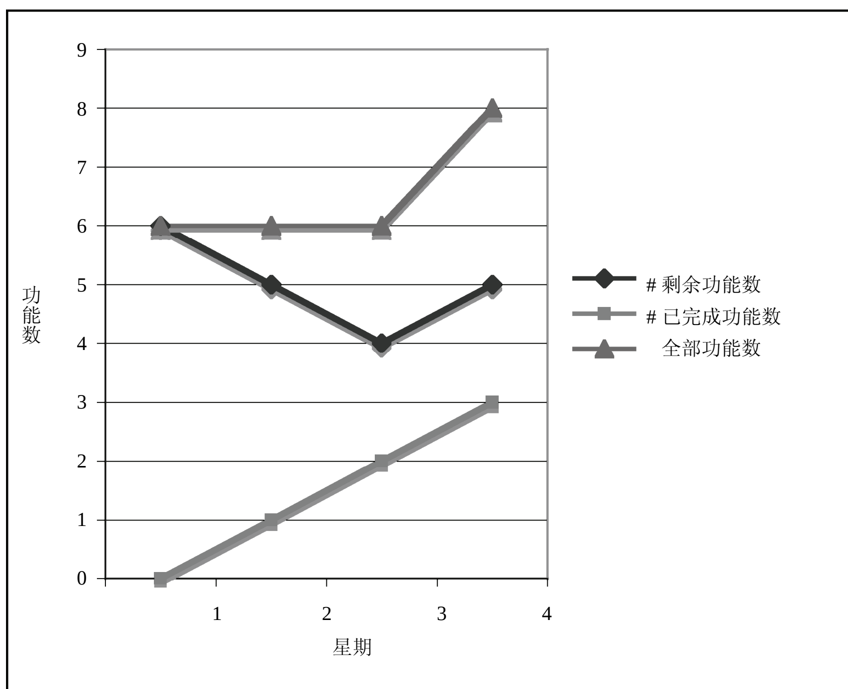
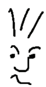


图11-3　六周的速度图表

客户见到团队已经完成的工作后，说道："看起来不错，可我现在真地希望能在这里加入这个功能，在那里添加那个功能。"

"这个"和"那个"功能每个要花费团队另两周的时间。项目经理原先的计算认为，你们已经在60%的时间内完成了60%的功能。一切都在正常轨道上进行。而现在离轨道可差得远了。但是你提早获得了客户的反馈，而不必等到项目结束，这样你就可以为客户提供他们真正想要的东西。


小乔爱问……

为什么我们没有已完成的工作?

你已经在项目中投入了几个月的努力，也没有人偷懒。可是当你试图绘制速度图时，发现你们没有任何（或是几乎没有）已经完成的工作。怎么可能呢？这是有可能的。如果使用顺序式生命周期，或者用其他生命周期，但是按照架构进行实现、没有计划好如何完成功能的话，那可能性就更大了。

如果项目团队使用顺序式生命周期或是按照架构进行实现，他们就会产生很多部分完成的工作。在精益社区中，部分完成的工作被称为“浪费”，因为它们没有完全完成。速度图展示的是已完成的工作，项目经理可以了解到团队是在制造浪费，还是在开发完整的产品。

如果更多地使用增量式，甚至是敏捷技巧，你的速度图就会更多地展示出取得了哪些已完成的工作成果。让项目团队看到完成了哪些工作，这对维持项目节奏有好处，而且能让人们取得更多成果。

现在你获得了多少价值？我真不知道如何回答这个问题，因为这无法解释下面的事实：客户意识不到他们想要的功能在项目所分配的时间内完成是不可能的。最开始的度量方式并不正确。项目是有一些价值的。也许到目前为止，项目的价值要比项目经理想象得多，因为客户提早了解到之前所提的需求不太准确。但是项目的完成度恐怕就不是60%了，而是其他的什么百分数。

有些组织喜欢使用“完成百分比”。我也不赞成这种说法。这经常只包括开发工作，而未将测试涵盖在内。没有经过测试的产品功能就不能算完成。使用“完成百分比”会导致人们实施诸如6.14节中提到的日程游戏。

如果项目经理想了解真正的进度，可以使用速度图展示正在测试的功能。速度图能够揭示团队的真实进度，而不是计划进度。它还能展示出项目发生的变化以及正在发生的变化数目。

所以，跟挣值说不吧。用速度图取而代之。

11.2.4　用EQF跟踪最初的估算

Tom DeMarco在［DeM86］中描述了估算质量因子（Estimation Quality Factor，EQF）的度量方式。EQF可以使项目经理了解最初的估算质量。在项目中每隔一段时间，团队就要回答这个问题：“你们认为项目什么时候可以完成？”团队就项目预期结束时间达成一致，这个预期时间就是一个数据点。项目结束后，可以从发布日期到项目启动日期之间画一条直线。看看图11-4这个例子。你所绘制的线和“我们将何时完成”的线之间的区域反映了团队估算的准确性。这是一个

很好的技术，人们可以用之来提供反馈，检查个人的估算。即使不将其用作反馈，项目经理也可以用之观察项目实际发生状况。总之，这是一个很好的技术。

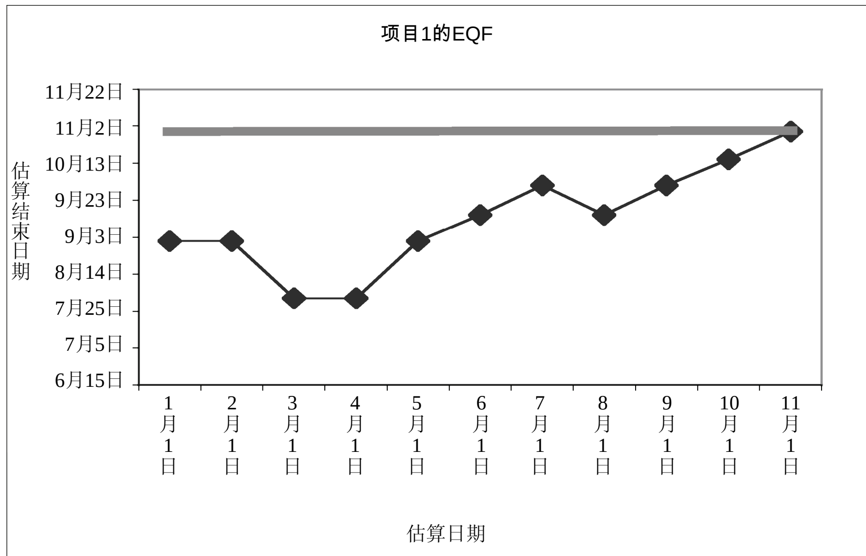


图11-4　估算质量因子

也许你在想：EQF不利于后期发现新的需求。没错，EQF不是完美的度量指标。不过，要是项目经理没有使用敏捷生命周期，迟来的需求（或后期搞清楚的需求）的确会带来影响。我宁愿了解项目延迟的原因，而不愿对其一无所知。

如果使用敏捷生命周期，项目经理的速度图会提供量化的答案，而不是定性的结论。要是没有用敏捷生命周期，EQF是一种很好的量化度量方式，可以用来观测估算的准确性。

图11-4是一个项目的EQF图，该项目预计9个月完成。在前几个月，当项目经理询问团队何时完成时，他们说是9月1日。前几个月，他们都很乐观，认为可以提前完成。但是到了第5个月，大家认识到他们对需求理解得还不够。他们的发现改变了整个架构，并推迟了完成日期。接下来的几个月，他们仍然不确定何时完成。到了项目的最后三个月，他们发现：由于架构发生改变，带来许多当初未曾预料到的问题。作为一种依据进度图表的检查，评估EQF这种定量度量方式对项目经理和团队很有帮助。

EQF不仅可供软件项目使用。任何项目工作都可以使用这个技巧。我在写这本书的时候就用了它。你可以用其帮助开发人员（或是测试人员、文案人员等其他人），指导他们提升估算准确性。

汤米在开发一个功能，他认为自己用三周可以完成。他确定自己每周都可以交付一些“小石

子”任务。完成一个“小石子”任务后，他更新了自己对于该功能的EQF，见图11-5。他觉得自己运气不错，因为可以提前交付这个任务。一直到该功能开发中期，汤米都没有改变自己的EQF，即使他已经提前成功完成了该功能的前一半任务。

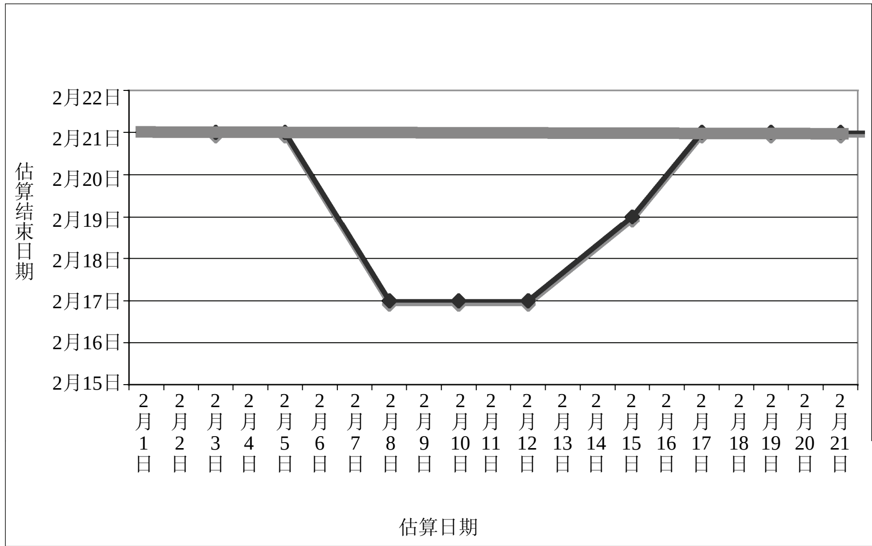


图11-5　汤米的估算质量因子

当汤米继续前进时，他就没有再像预先规划的那样取得进展了。虽然仍符合最初的估算，但是不可能提前交付这个功能了。

日程估算只是猜测而已，如果可以展示出实际进度与初始规划的差距，并解释其背后的原因，对于希望知道目前实际进度的人来说是很有帮助的。

11.2.5　更多的度量能提供更多信息

也许管理层只关心项目完成的度量数据，可如果你是团队的项目经理或是技术带头人，我想你一定希望尽早得到项目日程可能不准确的警告信号。为了保证了解项目的一举一动，我会监控如下多个数据。

- 除EQF之外的日程估算与实际值。如果使用速度图，其中会部分体现该数据。
- （有适当能力的）人加入团队的时间，以及实际需要他们的时间。
- 整个项目过程中需求发生的变化。如果使用速度图，其中会部分体现该数据。
- 如果没有使用敏捷生命周期，还要知道整个项目过程中故障反馈率（fault feedback ratio）

的变化。请参见11.2.9节。

- 整个项目过程中的缺陷修复成本，特别是在没有使用敏捷生命周期的情况下。
- 整个项目过程中，缺陷发现/关闭/未关闭比率值。

注意，这些是评估使用的度量数据，而不是试图找到项目中存在问题的度量数据。这些度量会暴露问题，但是仅凭它们不足以发现真正的问题。要把这些度量数据放在一起分析，才能发挥度量的威力。

11.2.6　如果只能度量项目日程，那就这么做

作为项目经理，你所在的组织也许一直都在使用顺序式生命周期，也许你刚刚加入某个项目，老板认为你的工作做得不能令他满意。这时，可以首先度量项目日程，看看到底发生了什么。

没错。注意我可没说只度量日程，我是建议这是你首要进行的度量工作。不过可别只用甘特图。项目经理有很多工具可以用。比如，可以看看团队应该在什么时候达成某个里程碑，再看看他们实际上何时达成该里程碑，如图11-6所示。要是团队启动项目时就已经落后了（无论第一个里程碑是什么），这个项目就不会在期望的结束日期完成。

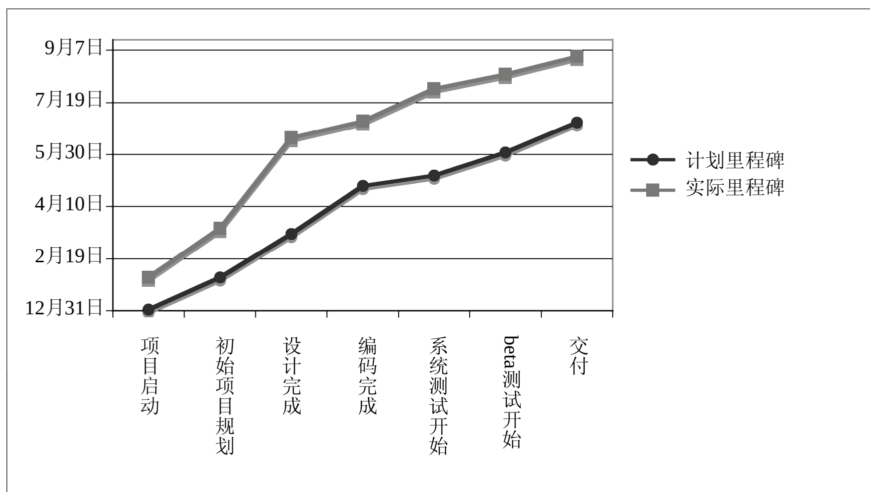


图11-6　估算日程和实际日程

失去的时间永远无法恢复。

图11-6中展示了一个项目发生的状况。这个项目使用了修改过的瀑布生命周期（下阶段工作的开始不必等到上个阶段结束），但是没有用迭代。注意项目的启动整整晚了一个月。当项目经

理贴出这个图时，他同时对管理层说道："别指望我们可以赶上一个月的时间。我们起步晚了，无法弥补失去的时间。"他对项目团队说："我希望你们抓紧时间工作，但是不要加班，也别把自己搞得太累。我们可没有容许大家犯错误的时间。尽全力做出最好的表现吧，我们会一直跟踪咱们的进度。"

作为一个讲求实效的项目经理，如果团队按时启动了项目，你可能希望让团队看到他们的进度状况如何。想看到这会产生什么样的图表，请看图11-7。

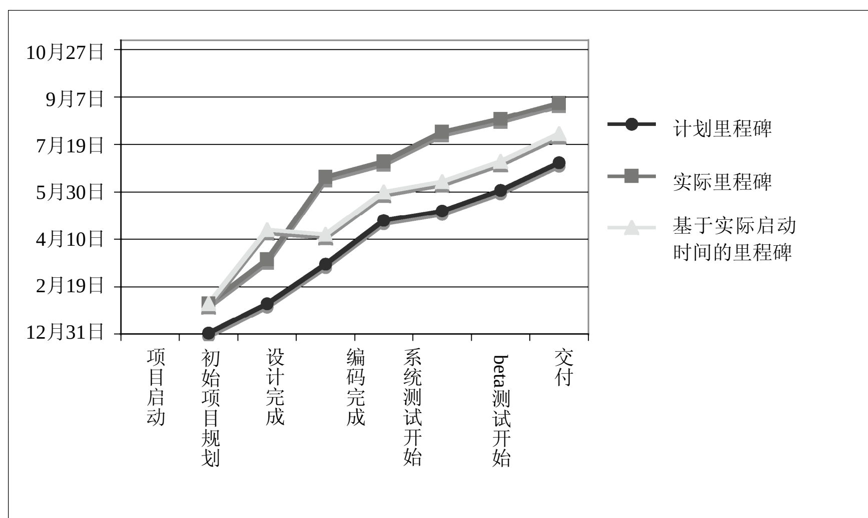


图11-7 日程估算与真实启动时间的实际日程

11.2.7 与项目团队人员分配情况相关的图表

项目在资源不足的状况下启动，这种情况经常发生。很多时候，项目最缺乏的资源是人。而且，人是软件项目取得进展的必备资源。然而不是任何人都行，这些人必须能够完成这个项目需要的工作。

打算仅用小团队启动项目，到了后期再加入新成员，这样做没问题，只要你做好计划。在此类项目中，计划就等于是在说："我们现在有这些人就够了，以后需要更多。"有些项目启动时缺少需要的人，与此类状况不同。我曾启动下列这些项目：有的项目在一开始就有做原型的人，而且团队中有技术带头人；有的项目需要我去要求开发人员修补缺陷，因为没有技术带头人；有的项目使用短迭代，而我只有开发人员，没有测试人员。这些项目中，我们都有计划，可以在项目需要其他人加入时，将他们顺利整合到团队中。

如果启动一个人力不足的项目，这就等于自寻烦恼。（如果项目启动时没有足够的计算机或

是办公桌等其他资源，只要团队可以处理好这些资源稀缺问题，这就可以接受。）项目经理要了解团队中有多少人，还得知道他们能否把工作做好。

要是项目经理面临类似状况，就使用带有时间盒限制的迭代吧，然后测量迭代的速度。你就可以把数据展示给团队，同时不断去跟那些仍不肯释放资源的人解释这些数据。如果总发生类似情况，请阅读7.7节，再想想这个工作是否还值得做。

图11-8展示了一个真实项目的人员变动历史。到了第2个月，此时项目需要的人仍然在做之前项目的工作（那个项目已经拖期了）。管理层没有继续等下去，而是让项目经理启动项目，他也遵命而行。到第3个月时，项目团队只有4个人，而不是所需要的10个人。这种状况下，项目经理和团队打算改变工作方式，却被告知其他的人可能“随时”到位，而项目经理和团队也信以为真。

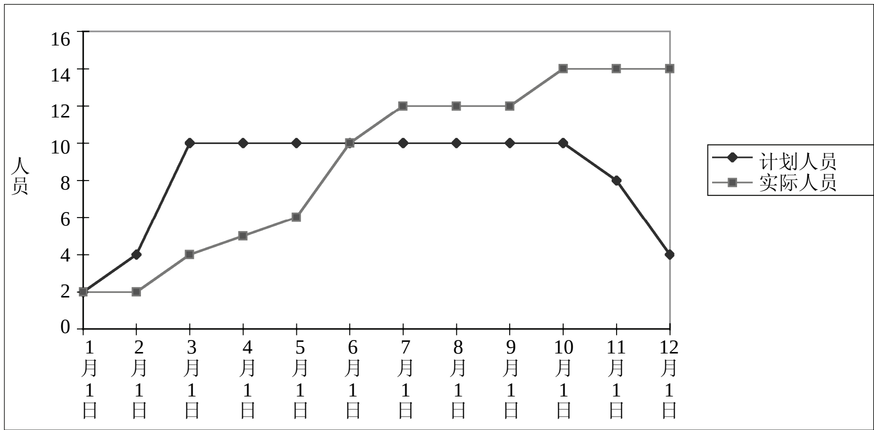


图11-8　项目计划人员安排与实际人员安排

到了第6个月，团队人员终于全部到位，可是进度却远远落后。为了赶进度，项目经理要求更多的测试人员，也如愿以偿。（该项目使用顺序式生命周期。）测试人员发现了很多缺陷，因此项目经理要求给予更多开发人员。于是开发人员创建出更多缺陷，测试人员又发现更多缺陷，最后，他们终于到达了一个相对稳定的点。

这个图度量了分配到项目中的人员趋势，而不是总数。如果把人员总数加起来，项目用到的实际人月要多出三分之一。人员成本不是这个项目的约束因素，所以问题不大。真正的问题在于：团队只交付了之前期望完成的三分之二功能，而且系统也不是特别稳定。

如果项目经理面临人员不足的状况，要小心不要陷入上面例子中项目经理所面临的同样状况，不要认为你不必重新设计项目计划。这个项目到最后还是分配了很多人，而下一个项目已经开始迫切需要人力了。但是项目经理从这个项目中学到很多经验。在他负责的下一个项目中，他

让两名开发人员按功能逐个实现，并使用为期一周的迭代。如果项目人员的到位状况无法达到当初规划项目时的要求，那就重新设计项目计划吧。对我来说，这几乎就等同于转向敏捷开发，因为它能让我在安排人员时拥有最大的灵活度。

11.2.8　判断项目的变化率

如果项目经理使用顺序式生命周期，也许无法绘制出有实际意义的速度图，因为你会被为数众多的未完成工作所淹没，同时又没有多少实际完成的功能（请参见11.2.3节中“小乔爱问”）。我仍然推荐使用速度图，不过项目经理需要将其拆分成几个部分。这种状况下，项目经理可以使用需求变更图，如图11-9所示。

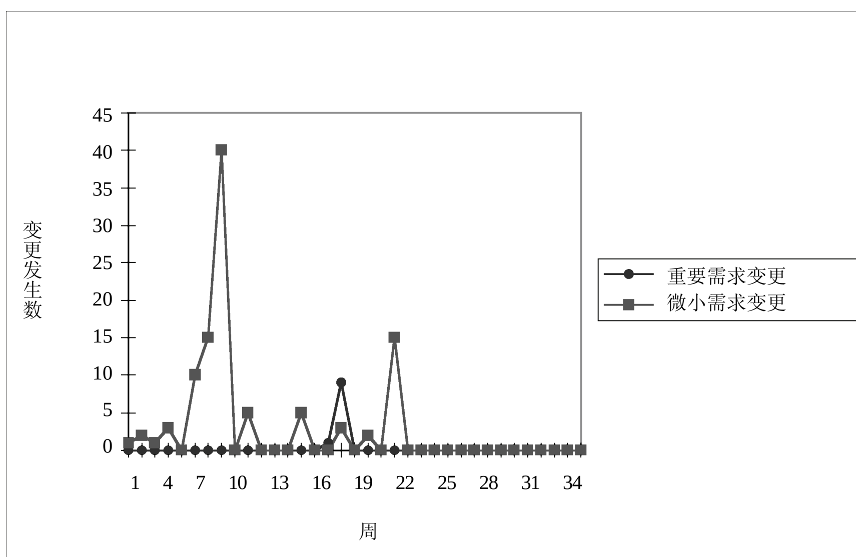


图11-9　需求变更图

我曾出任过一次类似情况下的项目经理。当时我中途加入了一个采取顺序式生命周期的项目，无法测量其速度。我们有太多没有完全完成的工作。不过我可以开始测量需求变更情况。在这个项目中，我有一个简单的标准，用来判断需求变更的重要程度：基于模块间接口的变化情况。因为这种变化很容易产生缺陷。只影响一个模块就是微小变更，如果多于一个模块受影响，就是重大变更。

在图11-9中，有许多微小变更，这也是我们希望在项目中看到的。不过到后面（第22周），也遇到一些重大需求变更。看到这些需求变化后，我就能向管理层解释项目可能会延期，或是缺陷数目会增加。有了记录的变更，我们就很清楚地知道：最初预计的交付日期、有少量缺陷数目的最初功能集合是无法达成了。

11.2.9　查看开发人员是在取得进展还是在白费时间

团队进入代码开发阶段后，项目经理就可以马上开始测量故障反馈率（Fault Feedback Ratio，FFR）了。FFR是指被拒绝修复的缺陷数（也就是没有真正解决问题的缺陷数目）与修复缺陷总数之比。经验告诉我，如果FFR高于10%，就说明开发人员遇到了问题，难以取得实际进展。

成功的敏捷项目中，开发每段代码都会用到测试驱动开发、结对编程和单元测试等实践，FFR会很低。而没有使用持续集成和持续代码复查的项目，其缺陷会不断累计，并欠下重大的技术债务（请参见附录B）。这时测量FFR就显得非常有用（如图11-10所示），从中可以了解到开发人员实际解决了多少缺陷，并与被拒绝修复的缺陷数做比较。

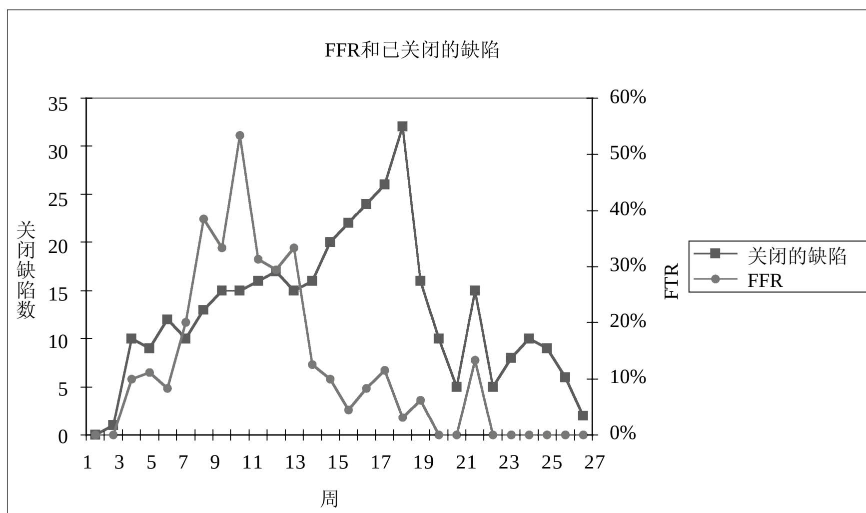


图11-10　缺陷反馈率

在图11-10中，当开发人员使得FFR降低后，成功解决的缺陷数就会增加。当FFR居高不下时，项目经理可以确定：开发人员们修复的缺陷数会减少，除非他们可以修复那些阻止他们前进的缺陷。

测量FFR可以每周一次，并可与开发人员和测试人员讨论该数据。如果看到某一周的FFR很高，先检查这一周修复的缺陷总数。如果修复了4个缺陷，有1个被拒绝了，开发人员跟测试人员可能遇到的问题不大。如果看到修复的20个问题中有5个被拒绝了（25%），很有可能是某个人或某些人遇到了麻烦。在图11-10中，可以发现在第6周前后FFR升高了，而且在第13周之前一直高于10%。一旦项目团队连续两周保持高FFR，项目经理就应该针对所有的修复开始同行复查（peer review）了。这会起效，FFR会降低到正常水平，缺陷修复不再会干扰正常进度，但是这个阶段与启动同行复查时刻之间会有时间差。

要想发现问题区域，不妨先问问开发人员，看看他们是否在修复时遇到了问题。一般我会这么问："你在修复这个问题时，是不是会在其他地方出现新问题？"我还可能问其他问题，最终都要问开发人员是否需要其他资源，以修复该问题。如果我听到开发人员希望重新设计某个模块，我们就会讨论这样做产生的影响。

接下来的问题是给测试人员的："你能否确定发生该问题的所有条件？"我从这些问题开始，看看开发人员是否能够一次解决问题的某一方面，或者了解测试人员是否足够了解系统从而进行全面的测试。

FFR是一种较迟的测量指标，项目经理要通过提问帮助团队解决问题。你可以在任何工作产品上使用FFR，不过人们不大愿意重新处理设计文档或是需求文档上的缺陷。没有代码，就无法测量FFR。因此，在顺序式生命周期中，越早开始测量FFR，项目经理就可以得到越多反馈。

11.2.10　测量查找和修复问题的成本

如果项目经理没有使用少于4周的迭代，团队查找和修复问题的成本就是一个关键测量指标。也许你已经见过类似图11-11的行业标准数据。

阶段	需求	设计	编码	测试	发布后
成本	1	10	100	1 000	10 000
项目1成本	未测量	未测量	0.5人日	1人日	18人日
项目2成本	0.25人日	0.25人日	0.5人日	0.5人日	8人日

图11-11　两个项目中修复缺陷的实际成本

这里是说：在需求阶段修复一个问题耗费1个成本单位，设计阶段修复一个问题耗费10个成本单位，编码阶段修复一个问题耗费100个成本单位，测试阶段修复一个问题耗费1 000个成本单位，发布后修复一个问题耗费10 000个成本单位。我们通常以美元或其他货币作为成本单位。

我曾测量过修复一个缺陷的成本，[1,2]发现这个数字在不同情况下各不相同。图11-11中展示

① 请查看http://www.jrothman.com/Papers/Costtofixdefect.html。
② 请查看http://www.stickyminds.com/s.asp?F=S3223_COL_3。

了两个项目的测量结果。项目1在出现问题时没有立即找到缺陷并移除，团队在代码中查找问题时三心二意。很多缺陷都是在测试时发现的。项目2从一开始就采取积极方式主动寻找缺陷。

要记住，重要的不是每个缺陷的成本，而是每个缺陷的成本乘以缺陷总数。要是不查看总成本，项目经理就不会知道该如何利用时间。基于之前项目修复缺陷的成本，项目经理可以判断是要采取积极手段，比如从一开始就查看关键项目文档、实施测试驱动开发或结对编程，还是只处理更棘手的缺陷，或者你可以监控修复缺陷的成本，并选择更为灵活的响应方式，比如同行复查修复代码或是检视所有的代码。

如果项目经理从未实施过积极的缺陷查找活动，找到一个缺陷的成本就会比较低，而修复一个缺陷的成本就会很高。修复所有缺陷的总成本将会非常高昂，因为如果团队中没有积极寻找和修复缺陷的文化，那一定会产生很多缺陷。如果一直在使用诸如测试驱动开发、结对编程、代码检视、同行复查等积极手段，那么找到一个缺陷的成本可能相对较高，因为已经付出了寻找缺陷的成本。但如果团队可以及早发现缺陷，修复一个缺陷的成本将会大幅降低。缺陷的总体数目也会减少，对于项目的发布来说，修复缺陷的总成本也就降低了。

我会监控寻找和修复缺陷的成本，这样就能了解代码库中的代码是否会让开发人员或是测试人员感到吃惊。我有如下一些基本原则，同时假定开发人员不曾主动寻找缺陷。

- 开发人员修复一个问题所用时间越久，他们就越有可能不敢碰触系统的某些部分。这很有可能是因为开发人员对系统的某些部分不了解。
- 测试人员找到问题所用时间越久，他们就对产品了解得越少，或者他们测试产品的手段就越匮乏。

修复一个缺陷的总成本越高，项目经理就越有必要管理缺陷相关的风险。比如，为了发布某个版本，何时停止接受修复，以及要修复哪些缺陷。

11.2.11　了解开发人员和测试人员是否在解决缺陷方面取得进展

绝大多数项目都会测量缺陷趋势。我曾见过一些很复杂的缺陷趋势图表，不过我最看重三个指标：每周发现的新缺陷数、每周关闭的缺陷数、每周仍未关闭的缺陷数。如图11-12所示。特别要说明的是，我不会在图表上标明缺陷的优先级，因为这会使得团队和高级管理层非常愿意玩出“缺陷提升和降级的游戏”（参见15.4.2节）。而且，无论缺陷的优先级有多低，开发人员也必须要把所有缺陷从头看到尾。所以我只要统计所有的缺陷就好了。

我计算尚未关闭的缺陷数目，是为了知道缺陷关闭率何时可以超过缺陷发现率，此时尚未关闭的缺陷数目就开始减少了。我希望看到尚未关闭缺陷数目曲线的拐点，当这条曲线的斜率变成

负值的时候，当前发布版本的风险也就减少了。

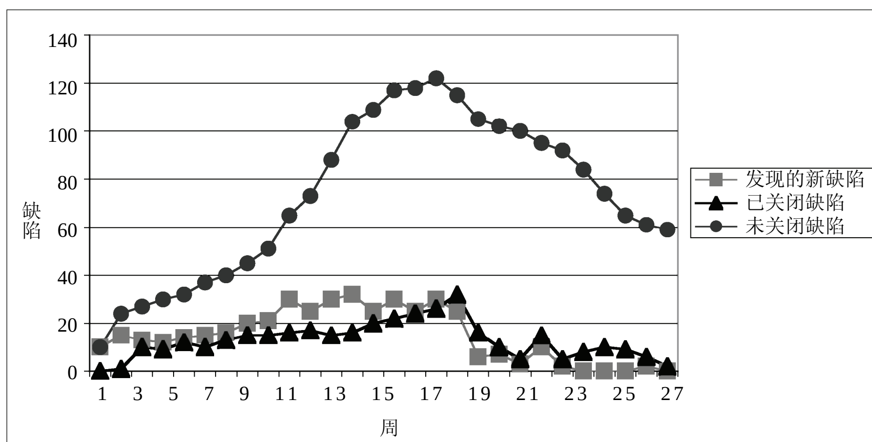


图11-12　一个项目过程中的缺陷趋势

11.2.12　展示测试过程

项目开始测试后，项目经理就可以马上绘制测试进度相关的图表了。无论你采用何种生命周期模型，我都推荐将测试与开发结合在一起。请看第13章。

在图11-13中，你可以看到规划的测试数目在不断增加，这是因为需求在不断变化。从中还可以看到，团队能够运行的测试数目在稳步提升，运行通过的测试数目也是如此。不过，在运行通过的测试数目与能够运行的测试数目之间有明显差异。

该图来自一个真实的项目，其目标是要发布带有新功能的新版本系统。在开始时，团队已经有了900个可以运行的自动化回归测试。可是当测试人员开始工作时，只有600个可以运行通过。这是因为团队没有使用持续集成，而是从一开始就采取了按阶段集成方式。当他们看到图表之后，意识到给自己增加了这么多额外的工作，他们就开始转向持续集成了。

测试进度图是用来衡量测试进度的绝佳手段。不过它也可以展示出（可能是不可预知的）需求的变更情况，同时也能从中看到开发人员集成工作的效果好坏。

如果采用敏捷生命周期，那么针对功能的测试就会在开发这个功能的迭代中进行，项目经理也许就不再需要上面这个图了。要是项目团队无法在迭代中完成测试工作，这个图还是会有帮助，特别是按迭代进行绘制的时候。如果使用迭代并且在迭代中开发功能，在迭代的开始就会看到“规划的测试”数目产生跳跃式增加。

11

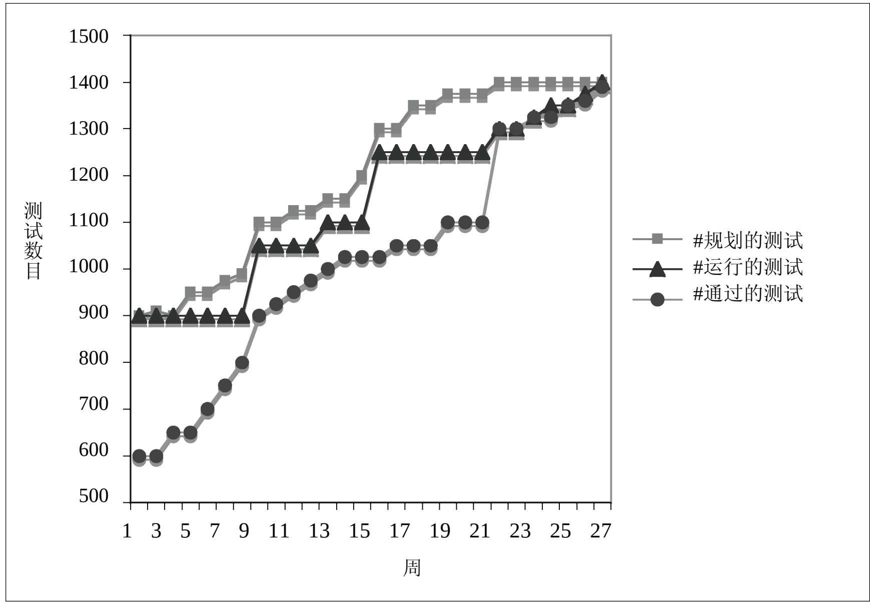


图11-13　测试进度图

11.2.13　展示定性数据

如果项目所有的数据都可以通过趋势图展现出来，那事情就容易多了。不过项目经理需要另外一种图表，特别是试图解释某种状态的时候。

功能、区域、模块	上次测试时间	状态	计划下次测试
功能集合1	3月3日，构建版本145	通过	构建版本150
性能场景3	3月1日，构建版本140	失败，等待杰夫和安迪修复	构建版本150
功能14	3月2日，构建版本142	通过全部回归测试	构建版本147，供用户验收
总体状态	到3月3日上午10：08	当前迭代开发进展至中途，看起来在按计划进行	当前迭代最新构建版本：构建版本165

图11-14　测试仪表板

当试图解释算法研究、性能场景和测试[①]等工作的进度时，我一直用如图11-14所示的进度图表，特别是针对时间较长的迭代或是顺序式生命周期。

11.2.14　用图表展示团队同意采用的实践

在第9章中，我建议团队可以采纳一系列技术实践。当团队同意之后，项目经理可以发现：让团队用图表把实践展示出来会很有帮助。项目经理不要自己绘制这些图表。你的团队成员都是成年人了，别把他们当成小孩子。如果他们不去推行实践，项目经理要找到原因。这就是所谓“项目管理”中“管理”的含义。

很多时候都是因为有些东西妨碍团队执行实践。项目经理的职责就是移除这些障碍。可以让团队在回顾会议中用图表绘制出这些实践（请参见8.2节）。

图11-15就是一个按阶段交付项目中实践图表的例子。为了取得项目成功，该团队选择了5个实践：伙伴复查、数小时内修复构建版本、按功能实现、波浪式规划、开发自动化冒烟测试。团队在伙伴复查、按功能实现和波浪式规划都做得不错，但是开发自动化冒烟测试和数小时内修复构建版本这两项就不怎样了。项目经理看到这个图后，不要去强迫让人们做得更好，你的职责是要发现妨碍他们成功的原因。

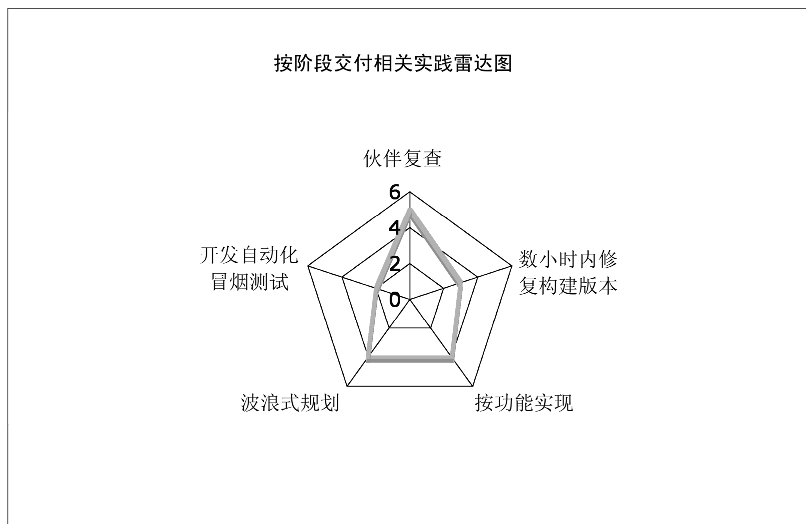


图11-15　按阶段交付相关实践雷达图

蒂娜看到这张图后，她召集了一次项目团队会议，并将解决实践相关问题放入会议提纲。蒂

① 请查看http://www.stickyminds.com/s.asp?F=S7655_COL_2。

娜发给每个人即时贴和笔，说道："有些东西阻止你们开发自动化冒烟测试和修复构建版本。请你们写下来解决这些问题所需要的资源。每张即时贴上写一个主意。写完之后，把即时贴粘贴到对应着'自动化冒烟测试'和'修复构建版本'的两张挂图上。"

过了七八分钟，大家都完成了。蒂娜大声读出即时贴上的字。她让团队将类似的相关想法分组汇总［RD05］。对于这个团队来说，问题很明显都是相关的。让人吃惊的是，问题在于自动化冒烟测试使用的源代码控制系统。蒂娜将解决这个问题作为项目新的需求（而不是目标）提了出来，并让团队中能力最强的两个人完成该任务。几周之后，团队重新组织冒烟测试和构建版本，这样人们就可以独立签入他们新增和变更的代码，同时不会影响到别人。

如果蒂娜不去询问团队障碍何在，恐怕她永远不会知道关键问题是什么。

图11-16展示了一个敏捷项目中的实践雷达图。[①,②]项目经理也许会觉得这个图中的数据有点奇怪。人们怎么能认为自己在测试驱动开发上做得很好，可是在100%单元测试开发上做得很差呢？当查理在团队回顾会议上提出这个问题后，马里奥解释说："是这样的，我开始时用了测试驱动开发，后来过于兴奋，就忘记要先写测试了。我给自己在测试驱动上打了高分，但是在100%单元测试开发上打了低分。我觉得我大概做到了98%。这可能还不错，但是有几次，如果我真地总是先编写测试，也许有些问题就可以更早发现了。"

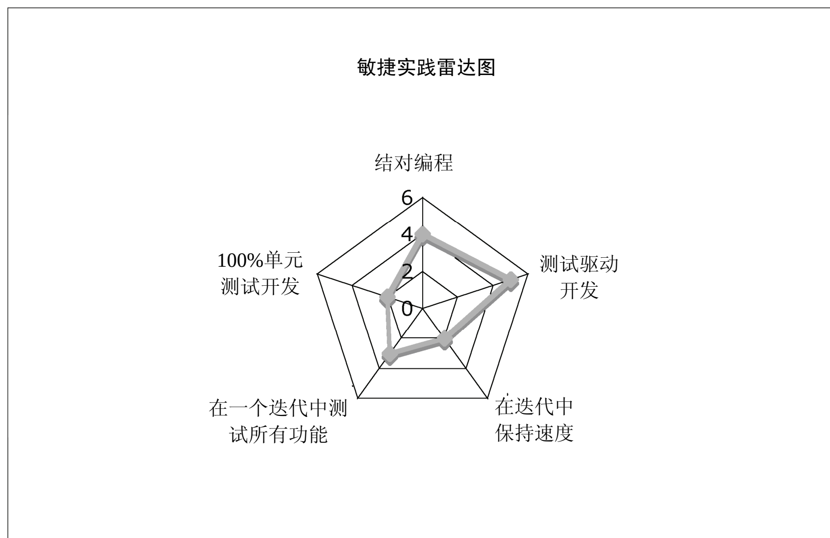


图11-16　敏捷时间雷达图

① 请查看http://www.xprogramming.com/xpmag/BigVisibleCharts.htm。

② 请查看http://xp123.com/xplor/xp0012b/index.shtml。

团队实践图可以让团队开放讨论他们选择的实践，哪些收到成效、哪些运转不畅。项目经理要知道如何阐释图中的数据，帮助团队发现问题的根源，并解决这些问题。

11.2.15　度量敏捷项目

如果项目经理选择使用敏捷生命周期，迭代时间长度不超过4周，同时在一个迭代中，所有分配给项目的人员不会被重新分配工作；如果你可以完成迭代中应该完成的工作，包括找到并修复当前迭代中引入的缺陷，那么你也许只需要速度图、测试进度图和迭代内容图就可以了。至于是否需要跟踪实践的运作状况，可以询问团队的意见。

11.3　为出资人创建项目仪表板

项目的出资人（或者其他需要听取项目状况报告的人）也许喜好某些特别的进度报告方式。但愿这些方式都能体现在项目经理的项目仪表板中。要是出资人不想了解任何情况，那么可以将仪表板上的数据反馈给项目团队。出资人希望知道项目何时可以完成。为了让他们知道这一点，项目经理可以告诉出资人项目有哪些风险，以及为了达成发布条件，项目目前的进度状况。

11.3.1　展示风险列表

项目经理在编写项目计划时就已经开始制订风险列表了（请参见2.3.9节）。风险列表会随项目进度发生变化。

在图11-17的风险列表中，列出了可能阻止项目团队达成项目驱动因素、约束和项目因素的事物。我也遇到过这样的出资人，他们希望像鸵鸟一样把头埋进沙子里，视风险而不见。不过大部分投资人都知道：视而不见毫无裨益。

在我的一个项目管理课程中，上课的同学们希望看看项目风险的分布图，如图11-18所示。他们发现，知道风险在图表中的分布很有帮助，这样就可以对风险有整体认知。我认识的高级管理层中，至少有一位希望先看看项目风险的分布图，然后再看下面列出的详细数据。

11.3.2　展示在满足发布条件方面取得的进展

查看项目满足发布条件的状况，可以让出资人了解项目的进展，如图11-19所示。如果他们不让你使用增量式或迭代式的生命周期，除非项目接近尾声，他们不会看到项目在满足发布条件方面取得多少进展。如果项目经理希望转向更加具备迭代或增量式特性（或者二者兼备！）的开发方式，就可以开始跟踪项目在满足发布条件方面的进展了。（没错，这是一种游击队式的流程

改进方式。）

风险 序号	风险说明	发生概率	严重程度	暴露程度	发生日期	应对计划
1	露辛达和她的手下无法在我们需要的时候参与原型审查	高	高	高，高	5月1日	1. 向露辛达说明时间表 2. 及时向露辛达告知项目进度 3. 提前1周提醒她
2	如果在9月1日之前发生变化，“供应链”相关的功能会改变整个数据库的设计	高	中	中，高	9月1日	继续与露辛达沟通这些改变造成的干扰。告诉她一个我们可以接受改变的日期。

图11-17　初始风险列表

风险的严重程度

高　☆（二）　☆（三）

中　☆（四）

低　☆（一）

低　中　高

风险的发生概率

图11-18　风险列表分布

条件	状态	状态	状态	状态
可靠性条件1：运行13小时	构建版本121，失败	构建版本123，通过，但是发现缺陷x、y、z	构建版本125，通过	构建版本127，通过
复查所有在线帮助	构建版本121，模块A复查完毕	构建版本123，模块B、C复查完毕	构建版本125，模块D复查完毕	构建版本127，除模块G之外其他复查完毕

图11-19 发布条件

11.4 使用项目气象报告

有些情况下，出资人或是管理层（甚至是客户）希望每天都能了解项目的进展。这是“微管理”。人们只有在没有数据或是不理解数据的时候才会选择微管理。如果项目仪表板上的数据对于这些人来说难以理解，他们会不断向项目经理询问信息。要记住，高级管理层会从较高的层面思考问题，而不会深入过多细节。我曾用气象报告[①]的形式，向这些希望了解细节的人报告进度，但是不在项目仪表板的层面上。

很多时候，我经常见到管理层在最后测试阶段进行微管理，所以我见到的气象报告经常基于测试人员的数据。然而，如果你的上司希望“微管理”你的项目，除了项目仪表板之外，再生成一个可以纵观项目全局的“平衡计分卡”气象报告吧。

项目气象报告会评估对比项目的当前状态和它的应有状态。我见过有些项目经理使用红、黄、绿三色交通灯模型来指示项目状态。交通灯模型展示当下的状态，而且易于理解。但对许多项目来说，有很多细微的差别，是只有三种状态的交通灯模型无法表现的。

我也见过一些出资人，他们想知道项目何时转向正轨。“永远不可能”，这样的回答虽然准确，却不能为人接受。我也知道，有很多项目经理主动或是被迫地控制交通灯模型。在最后一周之前，项目的交通灯一直是黄色或绿色，到了最后一周，却突然变成了红色。

① 请查看http://www.stickyminds.com/s.asp?F=S10522_COL_2。

一般来说，项目会按照团队现在的前进方向走下去，除非团队或项目经理采取某些行动改变方向。气象报告模型假设：除非项目经理和团队采取行动，否则天气不会转好。想要人为操控气象报告比较困难（但并非不可能），因为其中包括很多状态。

大多数高级管理层希望每周了解项目状况，他们不想费心琢磨你的项目仪表板，而是想让项目经理快速评估项目状况。而且这个评估要能让你和你的听众了解项目进度和未来进展。

气象报告能够让项目经理迅速掌握项目的全景，包括项目实际状态和计划状态，也就是项目的“天气”。如果“天气”一直在不断恶化，而且没有任何改善行动，大家都能看得出来：项目情况会变得更差（预测未来的项目天气）。

11.4.1　要小心定义气象报告的图示

由于项目状态的交通灯模型没有足够的状态标识，项目经理要明确定义不同的气象报告图示对组织的含义。你可以根据组织需要调整定义。下面这些定义适用于我的一个客户。


晴天：项目日程一切正常。


有少许云量：项目在日程上有点小问题，但是仍可以按计划进行。


多云：日程上有些问题，但是通过额外的努力，还是可以按计划进行。


阴：当前的日程或功能集合有很高的风险。


有雨：在项目目前的状况下，很难按计划完成，或是无法实现所有的功能。


恶劣：无论如何，我们都无法达成项目日程或是实现期望的功能集合。

为什么我推荐项目经理使用“有雨”和“恶劣”？“有雨”意味着“嘿，高管们，咱们得谈谈。我们无法完成你们想要的东西，不过还有机动空间。我们可以放弃某些功能，也许还能再加几个人（请参见7.5节）。延长一下时间表，或者再重新规划一下这个项目。”“恶劣”等于是在说：“我们搞砸了重新规划这个项目的机会，不过只要你们不中止，我们就会一直做下去。”

一个关于交通灯和气象报告的童话

本曾是一位项目经理，管着一个为期6个月的项目，其中问题层出不穷，不仅仅与项目相关。参与项目的20个人中，有一位突然结婚了，一位怀孕的开发人员必须有部分时间卧床休息，一位测试人员在滑雪时摔断了腿。但毫无疑问，这个项目的客户非常重要，客户希望得到更多的功能。

在最后3个月的时候，本将项目状态描述为“黄色”。项目还剩6周时间，本向非常重要的客户和管理层解释说，项目的几个新功能很可能无法完成了。没错，项目团队一直在努力工作。他们竭尽全力，希望按时交付，但是他们并没有十足把握。每次进度会议上，本的上司和非常重要的客户都会拍拍他的背，保证说他们对本很有信心。

本准备在下次进度会议上尝试使用气象报告。他将项目状态标注为“阴”。“阴？你是说完不成？”非常重要的客户怒气冲天。本的上司提议短暂休息一下，然后将本叫到一边，对他说：“本，你什么意思？项目状态一直是黄色的。你也总是可以让黄色状态的项目有所突破。”

本解释说，即使之前其他项目也是黄色的，但都没有像这个项目这么危险，难以完成。“我们真地需要点好运气。”

非常重要的客户询问本还需要哪些资源，并尽早在下周一提供他的实验室和测试人员供本调遣，希望这能有所帮助。本解释说这应该能起点儿作用，这些测试人员可以开始测试已经完成的功能，同时可以继续与开发并行测试。他们最后按时交付了，非常重要的客户提出的额外功能也只有一个没有完成。

在这个案例中，交通灯使得出资人相信本只是比较悲观，而没有认识到真正的问题所在。气象报告在这里协助出资人讨论真正的问题，并试图解决这些问题。通过改变进度的表现方式，气象报告也可以对你有所帮助。

假设你所管理的项目没有多少风险，而且一直按计划进行，你就可以给这个项目一个大晴天。

要是项目经理的风险列表条目与日俱增，而且你也不确定有两个功能能否准时完成。虽然项目仍然按计划进行，但是你确定将来一定会出问题。又好比现在还是项目早期，但是你的测试团队无法按计划运行测试，更不用提成功通过了。对于这两种情况，项目经理都可以将项目标识为“有少许云量”。这样的状态维持几周，开发人员和测试人员仍然不能取得进展，你就可以将项目标识为“阴”了。

如果项目风险列表在不断增加，开发人员虽然花了不少力气却是白费功夫，项目经理找到的缺陷也越来越多。这个项目就变成“有雨”状态了。

气象报告模型评估项目数据，以预测项目进度，项目就像是在面临季节的更替。预测来自于我们对于天气的经验，季节性的天气不会发生太多变化。即使有雨雪寒暑，天气总是在按着这个季节的规律变化。项目进度也是如此。项目经理也许会遇到一个可以解决的问题，可如果要是反复发现问题，你就得改变原来的评估了。随着天气图标的变化（或保持不变），看图的人会更清楚了解项目的状态，也许就会想要深入了解仪表板上的数据了。

11.4.2　建立可信的气象报告

如果气象报告每周总是发生剧烈变动，它就会失去了可信度，除非确实发生了某些严重影响项目的事情。一周之内可能影响气象报告的因素包括：相当可观的人员转而从事其他工作、厂商错过截止时间，或是没有认识到架构师无法为规划好的功能集合提供支持。

使用不太职业的气象图标也会影响可信度，尤其是在管理层非常关心工作外在表现的时候。项目仪表板应该让查看的人一清二楚，同样地，气象报告的图标也应该提升项目经理的可信度，而不是造成不良影响。

如果项目经理已经收集了大量项目数据，比如日程相关数据、速度图、缺陷趋势、测试覆盖率、人员分配、风险列表等等，那么气象报告就是评估整体状况的最佳方式了。要是没有收集这些数据，那也别光凭直觉产生气象报告，应该转而展示为了满足发布条件的项目进度。

11.4.3　每周发布气象报告

气象报告的目标是帮助人们了解项目评估状况，不让他们感到意外。还有多于两个月剩余时间的项目应该每周发布气象报告。在某些时间点上，比如项目状态仍在快速改变，或是接近某个重要的里程碑时，气象报告每周可以发布多次。

有了气象报告，项目经理的项目状态弹药库中就多了一件武器。可以将项目仪表板作为项目度量和状态报告手段的首选，不过无论使用何种方式，都要符合项目所在的组织及其环境。

铭记在心

- 使用速度图和迭代内容图作为首选。
- 数据是工具，不是目的。要记住，图表应该为你服务。
- 如果无法获取需要的数据，项目经理就遇到了比数据更严重的问题。先解决这个问题吧。

第 12 章

管理多地点项目

12

你所要管理的项目可不是七八个人坐在一起那么简单。实际上，你要同时管理多个地点的多个团队。这些团队可能分布在走廊周围、不同楼层、彼此相隔几英里的建筑物之中，甚至有可能分布在处于不同时区、拥有不同文化的国家里。欢迎进入地理分散式项目的世界：多地点项目。

如果项目分布在多个地点，单是沟通就会让项目变得异常复杂。项目地点分散越广，就越困难。不仅有时区上的差别，每个团队的文化也不一样。

也许你没有遇到全球项目；也许你很幸运，管理的两个团队间只隔了一个楼层。要认清现实，这也是一个地理分散式项目。只要团队中人与人之间的距离超过30米（大约32码），这就是一个地理分散式项目［SR98］。

也许你的团队没有跨越多个物理地点或时区，不存在文化上的差异，可是他们还是没有坐在一起。（如果部分团队成员与团队其他人要坐电梯或是走楼梯才能见面，这就是一个地理分散式项目［TCKO00］。）只要距离或时间的差异阻碍了人们的沟通，他们就算是被隔离开了。项目经理必须以管理多地点项目的方式来管理团队、文化差异和其他因素。

12.1　一个问题的成本是多少

科伯恩认为，即使是走到大厅去这样的行为，成本也非常之高。[①]也许你从来没有这样想过，但是分布在多个地点的团队确实让项目经理付出了代价。我们假设一个全心投入的开发人员每天可以挣到500美元，相当于每分钟1.04美元。那么这个开发人员提一个问题并找到答案的成本是多少？（其他团队成员的成本类似，但是团队中可能有多个开发人员，所以这个例子是以开发人员为中心的。）

① 请查看http://alistair.cockburn.us/index.php/Harnessing_convection_currents_of_information_060。

如果团队都坐在一起，这个成本就非常低。开发人员有问题，他们要么跟能够回答问题的人一起探讨，要么找团队以外可以回答这个问题的人。如果开发人员与能够回答问题的人一起工作，马上就可以得到答案。一个问题的成本为0美元。假设开发人员询问并非本团队的人；站起来，走过去，问问题，得到答案。这个过程要3分钟，其中开发人员走过去、问问题花费2分钟，对方回答要1分钟。总成本为3.12美元。如果团队一天有10个问题，那么每天就会损失30分钟，总成本为31.20美元。（我还没有计算干扰其他开发人员造成的时间损耗，请参见16.7.3节以了解更多这方面的成本。）

要是团队分布在两个楼层，有两件事会有所不同，问问题的成本逐步升高。首先，走路的时间变长了，因此开发人员要提问就会花去更多时间。要么他们就自己敷衍了事，不再请教别人。即使我们不计算没有问出来的问题的成本，总是会有些必须要问的复杂问题。要走到其他楼层才能得到答案，这些问题就得用更长时间才能解决。假设回答问题大概要用3分钟。提问的新开发人员会被干扰11分钟，而不是之前的2分钟。而回答问题的开发人员要被干扰3分钟。每个问题的时间成本是14分钟，金钱成本是14.56美元。我们假定开发人员每天问5个问题，项目每天丧失的时间就是70分钟，成本为每天72.80美元。

一周过后，丧失时间累计到350分钟，几乎6个小时，这都解决一个人日的工作了。这些时间的金钱成本为364美元。每一周，什么都不做，这个项目每周就丢掉了一个人日的工作。这里估算的时间损失还是保守的。在你的项目中，很有可能流失了更多的时间和金钱。

也许你会说：“噢，我们用即时聊天工具跟大家沟通，不用**看见**真人。”当然，有些问题可以通过即时聊天工具回答，但不是所有的问题都能如此，尤其是需要长时间才能回答的问题。

如果你是项目经理，要尽你所能让整个团队坐在一起。

12.2　识别项目的文化差异

如果所有的地点都处于一个国家，文化差异也许不明显，但确实存在。我曾管理过一个项目，其中两个团队，一个在波士顿，一个在洛杉矶。我们都属于同一家公司，但是两个团队实施的实践并不一样。两个团队都进行每日构建，但是在波士顿，大家每天都会签入代码（持续集成）；而在洛杉矶，他们只会在其他人需要使用时才签入自己的代码（按阶段集成）。当洛杉矶的开发人员被波士顿的开发人员搞蒙了之后，这个差异变得非常明显。洛杉矶的开发人员质问：“你明知这段代码有问题，怎么还能向代码库中签入呢？”波士顿的开发人员反击道：“你怎么能让我们一直等着看你的想法呢？”

同样的公司，同样的产品，对行为的期望却不同。这就是众多不同文化差异的一种体现。

文化差异体现在人们认为他们可以讨论哪些话题，体现在给予哪些奖励，体现在人们对待彼此的方式。上面持续集成和按阶段集成的例子，体现了职能经理制定的奖励机制和人们互相对待的方式。在洛杉矶，将“未完成”的代码签入到公用代码库中是不可接受的。在波士顿，要是代码在手上待了一天还没有签入，这是不可接受的。即使在同样的社会文化中，也存在着文化差异。不同地点的不同团队会养成不同的习惯——他们自己的文化。

项目经理在管理多站点项目时，要准备应对各种文化冲突。记住，整个项目不一定要使用同样的生命周期模型、开发手段和具体实践，但是整个项目团队必须使用互补的生命周期模型、开发手段和具体实践。

12.3　在团队间培养信任

多站点项目最重要的管理技巧，就是要帮助团队学会彼此信任。

如果多个团队拥有同样的承诺目标，同时他们的可交付物相互依存，那么随着他们完成自己的可交付物，同时朝着共同的目标努力，这些团队会学着互相信任。项目经理的工作是要帮助每个团队制定承诺目标，定义可交付物。管理项目日程和互相依存的可交付物有三种方式：确保每个地点都有完整的项目可交付物，确保团队可以彼此互相合作，帮助人们互相接触。

12.3.1　确保每个地点都有完整的项目可交付物

项目经理在组织项目时，要假定每个地点都会完成一些相对完整的功能集合。没错，可交付物是互相依存的，他们都会存在于最终产品中，但是每个团队都有责任交付一部分完整的功能。如果开发人员与测试人员不在同一个地方，那么项目经理很有可能因为无法得到可以工作的软件而失望。必要的沟通可能花费太长时间，或者根本无法进行。如果不同地点的团队功能相对单一，就会遇到这个问题。

除了不要创建单一功能团队之外，也不要创建过于“零散”的团队。“零散”的意思是指开发人员（或测试人员、技术文案等）分散在不同地区。如果一个团队有两个开发人员在芝加哥，两个开发人员在巴黎，这个团队就是“零散”的。如果人们散落在各个地点，他们就很难协作完成工作的交付。当事情进展不顺利时，人们一定会互相责怪。要是同一个团队的开发人员在一个地方，测试人员在另一个地方，技术文案在第三个地方，情况会变得更糟。

要保证每个地点都有一个完整的团队，他们交付相对完整的功能，而且必须是通过测试、可以顺利运行的功能。

如果无法在一个地点召集完整的团队，那就得想办法保证每个人跟团队其他人能够紧密接触。



小乔爱问……

开发人员与测试人员不在一个地点？

你的经理已经决定了，项目的测试人员要跟开发人员分开。作为项目经理，你该怎么做才能保证项目成功？

要把不同地点的人按照负责的功能进行分组。“你们三位来自密尔沃基，负责开发；你们两位来自班加罗尔，负责测试；大家一起完成这个widget的开发。”项目经理要帮助团队培养信任，并且知道如何共同工作。这很难。

如果你有其他选择，一定要拒绝这个方式。要在每个地点建立跨职能团队。

保持逻辑意义上的紧密联系

——盖伊，软件开发者，瑞士

我在销售部门工作，是唯一的开发人员。因为具备某些团队需要的技能，我目前正参与一个项目的开发。不过这里只有我一个人。团队在巴黎和伦敦还有其他人。

为了摆脱“孤军奋战”，产生更多团队的感觉，我们有一个开放的Skype连接，能够找到所有人，而且一直在线。每个人都有一个摄像头，大家都能了解互相的工作进度。我过了一段时间才适应一直在摄像头前的感觉。虽然还不完美，但总比啥都没有好。

我们一直保持着逻辑意义上的紧密联系，因为无法做到真正的紧密联系。这的确有助于我交付工作。要是没有这些联系，想完成工作就太困难了。

互相依存的团队之间不能存在竞争关系

我曾在一个组织中工作，这个组织想要降低软件开发的成本。他们认为降低成本的最佳方式，就是使用世界其他地方的低成本开发人员。他们有4个横跨欧亚大陆的软件开发团队。欧洲的经理被告知：如果他们不能及时交付，那么所有的工作就要被移交到亚洲去。

欧洲的经理一点都不傻。首先，他们在选择功能时精挑细选，确保自己能在短期内完成这些功能。在与他们第二次会面以切分功能开发任务后，亚洲的经理认识到这一点，并且也争得了一些不那么难以完成的任务。

接下来，欧洲的经理鼓励他们的架构师（仅仅）在白板上做设计，然后用相机拍下来，尺寸又小还看不清楚，再用邮件发给亚洲的开发人员。欧洲的架构师们执行了企业的要求：“将架构的副本发给所有地点。”可他们却忽略了这个要求的目的。即使图片的尺寸变大了，亚洲

的团队仍对架构设计迷惑不解。

项目的缓慢进度让公司副总担心不已。在跟经理谈话后，我告诉了副总一位经理说的话："如果你们坚持把我的开发人员置于失业的危险境地，我就不会帮助亚洲的项目团队。我会确保完成必要的工作，但仅此而已。你想降低开发成本？那就别把我们敌对起来。"

所谓"当局者迷，旁观者清"，根本原因在于大家之间已经完全没有信任感了。让团队互相竞争，这等于置他们的工作于必败的情形。信任被摧毁了，人们也被鼓励先考虑自己（而且仅仅考虑自己），项目的产出也因此而减少。有些时候，产出多少还有点；而有些时候，人们会破坏已经完成的功能，因为他们正在"修复"某些东西。

要确保多地点的团队有同样的目标——以项目或是工程为重。绝不要让他们彼此竞争。

12.3.2　确保各个团队可以互相协作

作为多地点团队的项目经理，确保项目的运作机制让团队互相协作是你的分内之事。如果团队开始互相竞争，特别是在他们都不想失业的情况下，你恐怕很难对外交付任何完整的产品了。

12.3.3　让人们建立个人接触

要是经济上允许，可以让全体团队成员到同一个地点一起工作一段时间。除工作之外，还可以组织一些社交活动，比如一起进餐，这可以让不同地点的人们互相增进了解。此处与7.2节中提到的强制娱乐活动不同。有了这样的经历，团队回到各自的地点之后，他们之间的沟通会更顺畅。

实在无法让团队聚到一起，那就让项目经理和技术带头人聚到一块儿吧。人越多越好。即使每个地点只能来几个人，大家互相认识、彼此了解，经过几天的工作后，大家也会更加信任对方，而不是像从未见过面的人那样彼此猜忌。而且，由于每个人都已经见过面，不妨继续组织各个地点之间定期的互相差旅访问，以维持彼此之间的良好关系。

多地点团队的项目经理应该定期访问各个地点。你大可不必每周或是每个月都去所有的地点转一圈，但是一定要在各个地点都呆上一段时间。我喜欢在每个地点都主持召开项目/工程团队会议，并让人们看到我。这样的话，即使我回到自己的办公室，他们还是对我有印象。

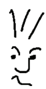
只要项目经理能够开始建立信任感，多站点项目的成功几率就大大增加了。但如果项目经理在启动项目时没有考虑这一点，那项目要想成功就需要许许多多的运气（还得发生一些奇迹）才有可能了。

人们需要面对面接触，这个过程中对彼此建立起来的感觉，可以让他们在即时聊天工具或邮

件上的沟通更有效率。假如你在电子邮件中看到这样的问题："你这周有任何进展吗？"可以做个实验，强调不同的单词。如果强调"任何"，人们可能感受到挖苦或是无可奈何。如果强调"进展"，人们可能感受到无可奈何或是焦虑。如果强调"你"，人们可能感受到的是来自个人、项目团队或你所在世界的问候。人们离得越远，就越有可能对你产生误解。他们可不知道你的话是在什么情形下问的。

12.4　在团队间使用互补实践

互补实践是指项目的生命周期，以及团队使用的任何开发、测试或是其他管理实践都应该是可以互补的。让一个团队使用瀑布生命周期，在最后交付一切，而让另一个团队使用迭代生命周期，期望定期获得反馈，这种做法行不通。



小乔爱问……

我有权要求每个团队使用特定的方式工作吗？

可能你认为自己没有权力要求团队使用互补的生命周期和实践。你当然有这个权力，而且必须这么做，因为你是项目经理。要是你无法要求人们以特定方式工作，那就要求结果吧。"我不管你们怎么做，但是这个功能要在两周之内完成开发、集成和测试，这样曼彻斯特的人们才能用得上。"

12.4.1　使用互补生命周期

首先要确保每个子项目团队在使用互补生命周期。对于多地点的项目，不要使用顺序式生命周期。因为要想尽早获得项目的反馈很困难，而且早期的里程碑很容易敷衍了事。在多地点项目中使用顺序式生命周期，很容易引发日程安排游戏，正如6.13节中所展示的。

在我指导过的成功项目中，有些项目的整个周期中每个团队都使用阶段式交付生命周期；有些团队在早期使用迭代开发原型并获取反馈，然后再转向使用某种形式的增量式开发。在一个项目中，我们使用了一个为期8周的迭代，称之为"原型探索"。每个项目团队都会将自己负责的主要功能原型化。接下来，我们用2周的时间评估原型以及在这些原型中浮现出来的架构。然后，我们又使用了3周的时间盒以实现和测试各个功能，并在每个时间盒结束时进行反复的集成。这就是迭代式生命周期（原型探索部分）和敏捷生命周期（时间盒部分）的结合。

一位同事管理的项目包括使用2周XP迭代的英国团队、使用4周Scrum迭代的以色列团队，以及使用按阶段交付生命周期的加利福尼亚团队。他们必须很小心地管理谁在什么时候实现何种功

能（他们也确实用到了16.6节中的概念），而且这样做确实很有效。

如果一个多站点项目的全部功能都使用顺序式生命周期开发，我就不知道怎么样才能按时交付了，除非置缺陷于不顾。要是项目经理希望同时掌控发布时间、功能集合和缺陷水平，那就不要用顺序式生命周期。如果不能联合使用迭代和增量式开发，至少要使用迭代。特别是对于多站点项目来说，开发人员和测试人员需要尽快得到反馈。

12.4.2　详细说明每个团队的里程碑和交接物

不同项目的生命周期不必完全相同，但是各个小组的产出成果必须满足其他小组的期望。这种期望的满足反映了不同项目组之间的互补实践。

在组织项目日程的时候，项目经理要确保大家都认同一些关键词汇和里程碑的含义。

1. 详细说明每个词汇的含义

不同团队采纳的实践和项目词汇都会不同。对于“修复”、“验证”、“功能冻结”、“编码完成”这些词汇，很多团队都有自己的理解。即使大家使用同一种语言也有可能出差错。

我曾经管理一个二线支持小组，他们要支持一个全球的多站点团队。大家对于“修复”一词的理解就不一样。波士顿所在团队的工作是修复一线支持小组无法解决的缺陷，而且客户对时间的要求特别紧急。有两个处于欧洲的一线支持团队，他们总是出现同样的问题。这些欧洲同事向非常重要的客户反复承诺，要修复一些时间紧迫的缺陷。可缺陷的修复总是不完整。在波士顿团队看来，“修复”这个词意味着：缺陷已经调查过了，也找到了原因，并且修复方案正在接受测试；而“已验证”则是指修复已经完成——修复方案经过测试，能够真正解决问题而不会产生其他问题，并且这都是通过验证的。欧洲同事们认为“修复”就应该是说确实已经完成修复了，可波士顿团队却不会将“修复”视为最终状态，“已验证”才是最终状态。

有些团队发现：包含“冻结”或“完成”字样的里程碑很容易造成困惑，比如“功能冻结”、“代码冻结”或“编码完成”。（又一个不使用顺序式生命周期的好理由。）在我参与过的一个项目中，美国的开发人员认为“编码完成”是指首次冻结代码，以便于产生一个构建版本。而俄罗斯的开发人员觉得“编码完成”应该是指最后一次冻结代码，以产生最终构建版本，并生成最后的产品。在制定项目时间表时，两边的技术带头人一直在反复争论，直到最后，他们才发现两边使用的词汇含义根本不同。

2. 详细说明里程碑的含义

项目团队不仅要就项目词汇和里程碑是什么达成一致意见，项目经理还要保证每个人对里程

碑的含义都有相同理解。多年以前，我曾担任工程经理，试图将来自波士顿、洛杉矶和日本的项目产出组件组合在一起。技术带头人和我一直在制定项目日程。一开始进行得很顺利，直到遇到第一个里程碑：功能冻结，我们希望就这个里程碑的含义达成一致。对于波士顿团队来说，“功能冻结”意味着完成更为详细的设计。而洛杉矶的团队认为：只要对概要设计有了很好的解决方案，就可以算“功能冻结”了。洛杉矶团队希望系统尽可能具备灵活性，这样就不必过早急于向产品中加入功能，他们不能理解：为什么波士顿的人们一定要完成模块接口设计？而波士顿和日本的团队希望提早定义出功能，并尽早冻结接口，为日本的产品功能定制做准备。

我把团队技术带头人都召集起来，一起讨论各组需要什么，并定出交付时间。开始时，日本的技术带头人沉默无语，担心自己的观点会影响整个团队。我们回顾了项目需求：在同一个日历月内发布产品的英语和日语版本，并为英语市场创建公开的API。我们没有足够的时间将功能全部推广到日语市场。

我没有强制推行某种解决方案，而且让各个技术带头人谈谈他们各自面对的问题以及解决方案。波士顿团队需要及时冻结API，这样日本团队才有时间开发定制版本，也给文案编写产品文档留出足够的时间。洛杉矶团队需要创建足够的产品底层架构，这样他们就不必为后续版本修改API。日本团队需要修改GUI和数据结构，以适应日本市场。

如果波士顿和日本团队不能尽早定义功能，他们的开发人员接下来的工作就会很难开展。如果洛杉矶过早定义功能，他们的工作就很难开展。我们虽然开发同一个项目，但是各个团队的目标却不尽相同。一旦认识到这一点后，我们就能更好地说明要在什么时候需要哪些东西了。

作为同一个项目团队，我们将重点放在临时结果上（也就是每个组到底在何时需要什么样的东西），共同定义出了主要的里程碑。经过一周的时间，我们就每个里程碑的含义达成了共识，特别是与“冻结”相关的里程碑，以及我们如何知道已经达成了这些里程碑。虽然无法做到每个人都对整个日程满意，但是结果大家都可以接受。

通过想要的结果定义里程碑，这就是上面的技巧，我称之为“讨论并发布（discuss and publish）”。有些团队除了定义整个项目计划主要的里程碑之外，还要定义临时的过渡性里程碑。只要团队们都共织每个项目里程碑的含义，项目经理就可以制定联合项目日程了，并且可以搞清楚为了达成这些里程碑必须要做哪些事情。

3. 详细说明团队了解自己工作进度的方法

项目经理在管理多地点项目时，不要强制规定每个团队取得可交付物的工作方式。应该说明你想要的结果。已经成熟的团队通常会有一些成型的实践，包括工作产品复查、配置管理、产品构建、产品测试，还有其他实践。举例来说，虽然我个人非常崇尚每夜构建和冒烟测试，可我不

会要求每个项目组推动每夜构建。相反，我会将注意力放在详细说明想要的结果上，并管理达成这些结果可能发生的风险。我要求提供“大型图表（Big Visible Chart）”，并将它们发布在内部网上，让每个团队都可以访问。

假如你所管理的项目分布在三个地点：英国的曼彻斯特、美国的纽约和旧金山。你会面临很多由于时差造成的问题，不过团队带头人和你应该商定一个统一的时间来召开电话会议。你正在使用为期4周的时间盒来完成功能。在此之前，你曾与纽约团队一起工作，并且担心他们不知道如何在时间盒内完成测试。你需要一种方式，以帮助大家知道他们正在时间盒内完成工作。

由于你希望管理最后的结果，而不是人们已完成的工作，所以你可以要求团队带头人，让他们在每个星期的特定时间来解释测试进展状况。这样一来，你和特定的团队就可以知道：他们的测试是否维持着与开发同样的步调。你也可以让团队公布他们的测试进展，并放到测试仪表板中。请看11.2.13节，从中可以知道展示进度的一种方式。

如果团队负责维护大型图表，他们就会愿意去监控图表和自己的进度。

提示：管理结果

无论管理哪种团队，都应以结果为重，而不是方式方法或具体实践。这对于多地点团队尤为重要，因为项目经理无法以到处走动和倾听的方式进行管理［RD05］。

使用回顾来帮助人们反思实践，从而判断继续哪些方式、变更哪些方式。但是不要强制推行某种工作方法。要明确表述出你要的结果。

---

4. 详细说明团队如何评审工作产品

帮助团队就里程碑含义达成一致意见后，项目经理接下来就该考虑如何进行技术评审了。所有项目都会从评审中获益。如果人们无法聚在一起，项目经理应该提出一些评审的不同方式。（我从未见过能够跨越物理地域的结对编程实践。当然，也许某些时候可以起作用，但如果一个人处于某个地点，另一个人处于其他地点，我从未见过这样的实践可以长久实施下去。）

特别对于多地点项目来说，技术评审可以提供额外的沟通框架和上下文，人们可以借此机会讨论项目问题。有些人不喜欢谈论与整个项目相关的问题，他们可能更愿意谈论某些技术话题，而需求和架构评审可以让他们有机会提出疑问。要考虑采纳生命周期模型的不同，从而为项目的需求和架构评审安排合适的日程。如果有其他文档，也可以拿来评审，比如设计和功能规范，特别是代码。

因此，敏捷生命周期对于多地点项目尤其有用。该生命周期认为各种评审十分必要，还可以保证所有的开发相关工作（包括测试和文档）都能在时间盒内完成。

在多地点项目中，各地的人们甚至很少有机会进行非正式的结对。如果项目经理可以建立任何人都能使用的评审机制，那最终构造出来的产品一定更为牢靠。

在项目一开始，如果项目经理可以带动技术评审，不同的项目小组就很有可能针对自己负责的项目工作把技术评审继续进行下去。我会在项目初期评审项目章程、规划和日程表。即使项目经理不希望团队评审自己所有的工作，也应该在整个工程团队范围内评审工程的发布条件。问问他们，如何处理工程的发布条件，并将其转化成自己负责的项目工作的发布条件。

项目经理要鼓励团队成员评审自己的工作，这样可以培养出积极的氛围，鼓励大家互相评审。这种环境对任何项目都很难得，尤其是多地点项目。工作产品评审进行得越多，人们就更能深入理解自己的工作如何融入到整个产品中，并进一步搞清楚产品的工作方式。

搞清楚时差对于团队流程的影响

多地点团队刚兴起的时候，很多资深经理人都在想："我的团队分布在全球，他们可以让我的项目在一天24小时中都取得进展。"

不要轻易下结论。如果你的项目不需创新，而且对成本控制也没啥要求，那么散布在全世界的团队也许"可以"取得一些进展。

如果项目需要创新，就必须让项目团队成员更易于取得联系。假如处在圣马迪奥的丹在上午11点有个好想法，并希望与处在班加罗尔的维哈亚分享这个好主意，可维哈亚所在地的时间是当天半夜12点半。丹和维哈亚希望找到一些讨论的时间。丹不能在那个时间给维哈亚打电话，他必须得等，这就花费了项目的时间和金钱。请参见12.1节。

版本发布的时间越紧，时差对项目的负面影响就越大。如果项目经理要管理分散在全球的团队，那就转而使用短迭代吧，每个团队成员负责自己的可交付物，这样他们面对的问题也更少。

项目中要是面对很多时差和文化上的差异，多地点的开发工作就会很难进行。

项目经理有可能发现：相对其他项目，多地点项目需要更为正式的需求和架构评审。正式的评审有助于降低沟通问题产生的风险。只有在正式的评审机制下，有些人才愿意发表意见，而有些人也只有在这时候才能意识到项目经理希望他们发表意见。就算人们愿意提出自己对需求和架构的想法，如果项目经理不在项目中留出时间，来自不同团队的人们也无法做出评论。毕竟，在多地点项目中，你不大有机会在咖啡店中遇到别人，聆听他们的想法。

使用正式的需求和架构评审，项目经理可以确保团队每个人都能了解项目的目标。每个项目团队至少出一名技术代表参加正式评审。这些代表要能够就需求的正确性达成一致，并认可需求的可实施性。

对于使用迭代和敏捷生命周期的项目，我要求只讨论当前迭代的需求和架构。在使用增量式生命周期的项目中，我要求讨论仅限于当前阶段的可交付物，当团队需要进入下个阶段的实施过程时，再讨论那个阶段要交付的东西。要让团队重点评审接口，而不是功能的内部实现。

评审和考察世界其他地方（即使仅相隔30英里）的文档并非易事。如果只用邮件评审及考察需求和架构相关的文档，我曾碰到下面这些问题。

- 首先阅读工作成果的人会“主导”讨论。有些人不习惯通过邮件方式提出意见，这些害羞的人不会积极参与，因而评审的价值也就降低了。
- 如果已经有人开始提意见，有些人就不再阅读这些文档了，他们以为别人回去会处理这些问题。
- 想让大家形成同样的沟通风格很困难。如果项目经理能够在早期安排一次会议，让全体团队在同一地点参加，就有可能及早发现并解决这个问题。

那么这些问题在多地点项目中有什么不同？参与者所在的地区越多，时差越大，文化上的不同就表现得越明显，而且这些差异越难解决。人们喜欢用文档、邮件来暗自表示自己的不屑，而不是提出帮助和建议。如果项目经理希望解决问题，同时不想让别人感到受侵犯，就应该跟对方面谈。同样，时差问题会形成障碍。

文化差异（特别是人们要怎样才能觉得可以自由发言，并将重点放在要讨论的话题上）只能通过真正的谈话接触才能缓解。我喜欢面对面的讨论，要是难以实现，视频会议也是不错的选择。要是团队有过成功的经验，电话会议也许就足够了。以我的经验来看，人们在编写规格说明和提出意见时，他们使用和理解语言的方式会阻碍有效的邮件评审。要是团队大部分人的母语都是英语，而一小部分人的母语不是英语时，这种情况经常发生。我喜欢采用亲自与技术人员碰面的方式，与他们一起评审需求和架构文档。比起产品失败的潜在风险，我发现出差的成本要少得多。

如果项目经理管理的项目周期很短，而且愿意承担一些风险，不去满足某些潜在客户的需求，那就可以使用一些基于因特网的工具召开会议，同时结合高质量的音频连接。要确保会议主持者有熟练的会议推进技巧，特别是进行电话会议的时候。（请参见10.10节。）

12.5 寻找潜在的多地点项目和不同文化导致的问题

多地点项目面临自己的技术问题。它们的规模会更大，而且需要进行工程管理和项目管理。从我的以往经验看来，解决任何技术或产品问题的重要性，仅次于管理人与人之间的互动。在管理涉及多种文化的项目时，要注意下面这些问题。

- 不同地点的团队对里程碑和交接物的定义可能不同，这会让团队无法正确理解他们应达

成的承诺和交付给其他团队的工作成果。有时，定义的不同是因为缺乏对用词的正确理解。有时，人们对于承诺的含义理解不同——承诺，是指团队只是尽量去做、而不是拼尽全力，还是说人们应该倾其所能、想尽一切办法实现承诺？不管是什么原因，对里程碑含义的不同理解，可以通过一些互补的产品开发实践解决，典型实践包括：项目规划、项目日程安排和技术评审。

- 项目进度的沟通和报告出现不一致，经常在项目初期发生。看不到其他人在做什么，这会丧失对其他团队成员的信任。特别是对于具有不同地理区域和文化的团队，这种不信任会极大阻碍项目的成功。
- 对于彼此母语不同的项目团队成员来说，可能很难理解对方在说什么。语言上的差异，以及每个人对于项目通用语言掌握能力的不同，会在项目中产生很多问题。使用哪种语言作为通用语言？使用这种语言时存在哪些二义性？项目所有的参与者是否流利使用这种语言？是不是每个人都愿意与其他人沟通？是不是存在某些文化习俗，使得有些人不愿意与某个团队或某些团队成员沟通？要确保团队在书面和口头沟通中使用的语言适用于每个人。
- 节假日、假期和加班都有可能产生沟通问题。明确说明假期的含义、每个团队成员所在国家的法定假日对项目日程的影响，还有加班的概念，项目参与者与其他人的工作方式、项目进度的报告方式都会受到这些因素影响。
- 要确认每个人正确理解“当天结束”或其他某个时间对于可交付物的含义。比如，波士顿时间的下午五点，在加利福尼亚就是下午两点，在印度的时间又不一样了。
- 使用一些通用工具的水平不同。比如配置管理系统、缺陷跟踪系统、项目的内部网工具等。工具是为了鼓励共享设计、源代码、测试，还有其他项目信息。如果项目团队的某些人无法使用某些资源，他们可能会怨恨某些能够使用资源的人。而且，他们可能会不再与项目团队的其他人分享自己的工作。

中途回顾（请参见8.2节）有助于识别并解决这些问题，还能帮你发现并解决特定于你的项目的问题。如果项目经理无法召集所有人，可以在每个地点进行回顾活动，并将各地的项目经理叫到一起，解决跨地点的问题。

12.6 在外包时要避免下列错误

我本想将这一节命名为“如何成功实施外包”，但是我没有，因为我不能自称有过真正成功实施外包项目的经验。我曾管理过按时交付的外包项目，但是没省多少钱。我曾管理过节省了资金的外包项目，但是却没有交付客户真正想要的东西。我也曾拯救过许多团队深陷于缺陷之中的外包项目，这些项目当初的日程安排是一厢情愿，而最后的成本要比最初的预算高出200%到

500%。

如果项目经理必须在项目中使用外包团队，就要把下面这些提示[1]谨记心间。

- 训练外包相关负责人员。负责外包的人要了解产品的工作方式，包括产品内部、客户希望解决的问题这两个角度。
- 挑选厂商。厂商是否有相关领域知识？他们在财务上是否可靠？合同中是否包含知识产权相关的保护条款，以保证厂商不会侵犯相关知识产权？
- 选取最好的项目经理，作为外包项目公司内部的项目经理。当然，接包方也有项目经理。这个人需要跟你们公司的人沟通，以确保他们团队正确理解项目相关的可交付物和交接物。
- 与接包方的管理层或项目经理建立信任关系，这样可以帮你了解项目的真实状况。
- 规划、调整内部职员的工作时间，包括外包相关人员的时间，这样人们就可以找出时间互相沟通。如果你不能安排足够的人早点开始或是晚点结束一天的工作，地球那头的人要是有了问题，也许就找不到可以问的人了。经常发生这样的状况：相差8小时时差的工程师需要某些信息，而这边办公室里却没人，而且也找不到合适的人。这就无法做到全天候工作了，项目工作必须停下来，直到工程师可以找到答案。

  这对人们的要求可高了。你在让人们调整他们的工作时间，以便于与接包方的工程师们交流，而这些工程师有可能抢去他们的工作，或是已经抢去了曾在走廊一侧干活的同事的工作。别指望你的职员们接受这些，并能听候差遣——除非经济状况很差，他们也找不到其他的工作。
- 用文档记录需求。如果本地的技术人员都无法理解你对于产品的想法，又怎能指望远隔千里、英语非母语的人们理解你的需求呢？
- 制定适当的变更流程。尤其是开发工作遍布全球多个地点的项目，你需要一个清晰的变更流程，以确保只允许发生希望发生的变更。
- 选择需求相对稳定的项目进行外包。如果需求变更频繁，而且你总得跟用户确认不断演化的产品，那么分散在世界各地的开发会让这个过程变得更为痛苦。
- 为每个项目规划更长的时间、更多的资金预算，特别是在外包关系建立初期。我的经验法则是：为第一个项目增加30%的估算时间。然后再监控项目，看看是不是需要再增加估算时间。
- 在整个项目开发过程中，坚持接包方不变更团队。否则，花在训练对方团队了解你们产品的时间就浪费掉了，你还得重新开始培训流程。
- 为了支持接包团队，要确保有合适的工具、信息系统和流程。他们需要访问源代码、缺

12

① http://www.computerworld.com/managementtopics/outsourcing/story/0,10801,84847,00.html.

陷跟踪系统、数据库或其他平台应用、构建版本等信息和数据，要与内部团队用到的项目工具相同。

- 对于声称自己负责项目工作的人们，要验证确实是他们在做项目的工作。多年来，美国公司一直使用调包计获得合同。资深人员会先出面拿下当前的项目，然后转向下一个容易上当的傻瓜——呃，是客户，而这个项目的工作却由刚从校门走出来的学生们和其他没有经验的人负责完成。

你猜怎么着？美国之外的外包公司也学会了这一招。如果你无法验证谁在从事项目的工作，你的项目就很有可能成为接包公司人员提升简历工作经验的试验田。

铭记在心

- 如果团队没有分布在同一楼层的10米之内，这就是一个多地点团队。
- 相对于管理坐在一起的团队，管理多地点团队要花费更多时间，还需要更多的项目推进技能。
- 如果项目经理无法构建起与远程团队的信任关系，项目就不可能成功。

第 13 章

在项目中集成测试

13

测试不仅仅包括系统测试和单元测试。对于一个项目来说，项目经理要考虑很多种测试。如果产品中包括固件或硬件，或是要与其他系统集成，项目经理在安排单元测试时就要使用方法存根。也许在最终系统测试之前，项目经理必须先一起测试几个功能。每个项目有其独特之处，项目经理要想清楚需要哪些类型的测试，可以借助项目风险进行判断。

我建议，项目经理在一开始就做好测试的规划。如果你在规划时，不打算将测试与开发工作集成在一起，就没有人会去做测试了。到最后，项目经理就得把测试作为一堆独立的项目任务。这些任务启动得很晚，会耗费更长时间，同时会将任何类型的生命周期都变成顺序式生命周期，将项目拖得更长，同时延迟了开发人员得到反馈的过程。将开发与测试同时进行，可以帮助项目经理从项目一开始就集成测试。

13.1　从一开始，就让人们将“减少技术债务”牢记心间

这是某个新项目的第一天，项目团队聚在一起，整装待发。你也把一切都准备好了——除了需求之外的一切。项目团队的启动，可以从修复前一版本的缺陷开始。他们正在开发的产品也许是将要开发的产品的前一版本，不过这无关紧要。

在项目启动时，让开发人员和测试人员认识到在实际工作中走捷径的恶劣后果才是关键。

如果要进行后续版本开发，团队将会偿还之前欠下的技术债务（请参见附录B）。如果是开发新产品，项目经理就可以分配一些人构建原型，另一些人修复缺陷并以此了解当前项目中要避免哪些陷阱的诱惑。

要是项目经理决定先偿还技术债务，就要跟团队说明该目标。“我们在上个项目中做了很多技术上的权宜之计，这后来给我们带来很多麻烦。你们这次修复缺陷的时候，不妨看看我们可以采取哪些实践，以避免再发生类似的问题。”你甚至可以在团队从事新工作之前召集一次短期的回顾［DL06］，让大家分享和吸收修复问题时得到的经验。

13.2　使用小规模测试降低风险

你的项目并不那么好对付，面临着很严重的技术或日程风险。可如果在项目中使用的测试类型越广，就越能降低项目的风险。如果项目经理选择仅使用一种类型的测试，那就用单元测试吧。TDD（测试驱动开发）是实施单元测试的最佳方式。要是项目经理无法让团队先写测试再写生产代码，起码要让他们开发完生产代码后马上编写测试。

选择TDD而不是其他类型测试的原因很简单：就与测试的关系而言，TDD与设计的联系更为紧密。

TDD兴起于20世纪70年代［Bro95］。现在之所以能在项目中为人接受并得到应用，是因为它是极限编程（XP）的核心实践之一。不过不一定必须结合极限编程使用TDD，任何生命周期都能从使用TDD中受益。请查看13.3节以获得更多细节。

开发人员编写代码和缺陷

开发人员是要编写代码的。（他们还要搭建系统架构，进行系统设计，从而实现需求。）但是有些人忘记他们也会写出缺陷。如果一个开发人员某一天的工作状态不好，可能他写出的缺陷要比可以工作的代码还多［HT03］。

如果你的开发人员要等到项目临近结尾时才去查找缺陷，就会丢掉对他们来说发生最早而且最为便利的反馈：对代码的单元测试。不管是在编写代码之前（TDD），还是在写了几行代码之后进行，单元测试都是发现并修复缺陷的最简便方式。单元测试还可以用来阻止同样的缺陷一再发生（等到开发结束时再想阻止，这时的成本就很高了，请参见11.2.10节。）

即使使用今天最先进的编译器，开发人员也有可能发生编码错误。单元测试会发现编码错误，不仅仅只是那些由设计环境或编译器导致的问题。

也许你的开发人员会说："嘿，我们在使用一个非常高级的语言，所以不会发生内存泄露之类的愚蠢错误。我们不需要单元测试。"开发语言越高级，开发人员就越可能犯逻辑错误。逻辑错误的程度越严重，在整个测试过程中的系统测试环节就越难以发现。而使用单元测试找到这些错误就容易得多。

编写单元测试还有一个更好的原因，特别是对于使用TDD的开发人员来说。这些单元测试能够帮助开发人员设计系统，而不仅仅是发现缺陷。在编写生产代码前写几行测试代码，这样的行为可以让开发人员更清晰地了解如何编写生产代码。

相比于其他任何调试机制，单元测试能够以更低的成本帮助开发人员发现缺陷、设计系统。

13.3　TDD 是在项目中集成测试最简便的方式

作为项目经理，你认为将测试与开发集成是很好的方式，可是并不确定测试人员能否跟上开发人员的工作节奏。不过你很清楚，要是不把测试跟开发集成，项目就无法成功。此时，有这样一个解决方案。

TDD能够帮助你（项目经理和项目团队）集成测试与开发。实际上，如果项目经理没有专职的测试人员和专门的测试用电脑，使用TDD会显著降低项目中与测试相关的风险。

TDD遵循如下流程。

(1) 开发人员（如果是结对编程的话，就是一对开发人员）针对某个尚未开发的新功能创建一个测试。这个测试应该失败，因为功能的实现代码还不存在。

(2) 然后开发人员加入能够使测试通过的最简单的实现代码。

(3) 开发人员重新运行测试。如果实现代码能够通过测试，开发人员就重构以简化代码。如果代码无法通过测试，开发人员会继续修改代码。在修改时，开发人员会重构代码，使其变得更简单、运行更迅速、维护更简单——这都是为了防止发生技术债务。

(4) 随着开发人员继续工作，他们会运行所有的测试，以保证不会引入问题，使得回归测试不能通过。开发人员会继续遵循这个循环过程，直到所有的功能全部实现。

TDD是由内及外的开发方式。这是不是意味着就不能在一开始先做设计呢？不是的，不过先完成非常详细的设计毫无意义（也被称为big design up front［BDUF］），因为开发人员能够利用代码中的和谐之美（他们会在重构时发现这些美），并让设计根据自己的累积经验不断演化。根据我的经验，相对于一开始所做的前期设计，这样不断演化产生的设计（emergent design）会更简单。

采用任何生命周期的项目都能使用TDD。如果项目经理采用顺序式生命周期，则可以让开发人员使用TDD，从而减少在临近结束时发现过多缺陷的风险。当然，如果采用敏捷生命周期，并使用两周或更短的迭代，则需要使用TDD，这样就能在迭代结束时交付可以发布的软件，达成目标。

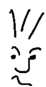
如果项目经理希望降低项目风险，不妨在风险最高的区域中使用TDD。如果希望增加最终产品的价值，不妨在价值最高的区域中使用TDD。不过要跟开发人员和测试人员讨论，看看是否要从风险最高或是价值最高的区域开始工作。请参见9.3.1节。

如果你打算在项目开始时探索原型，也许不需要TDD。不过我不建议这么做，尤其是在开发人员不愿意抛弃他们开发出来的原型的情况下。根据我的经验，先开发测试，可以驱动代码中算

法和结构的设计。开发人员对代码越了解，他们的决策效果就越好，项目风险也就越低。

单元测试不是万能药

即使单元测试是发现缺陷速度最快、成本最低的方式，它们也不是灵丹妙药。关键在于，开发人员必须对所有的代码进行单元测试，以获得其好处。你可以使用代码覆盖率工具进行测量。一般来说，即使是最有天赋、最负责任的开发人员，他们提供的测试也不足以完全验证他们所编写的代码。要是开发人员不把单元测试看成工作职责的一部分，那让他们编写单元测试还不如不写。项目经理也因此而会产生错觉，认为代码经过了测试，实际上根本没有。



小乔爱问……

如何让我的开发人员使用TDD？

看起来，TDD能够拯救我的项目。那我该如何让开发人员使用它呢？

首先，项目经理要保证不按交付日期衡量开发人员的工作。人们在进行新的实践时，要花更长的时间完成工作，因为他们所尝试的东西与以往不同。使用顺序式生命周期时，在编码阶段占用的时间会在测试阶段补回来。不过要确保开发人员在日程上有一定的灵活度，毕竟他们在尝试一个新的实践。

其次，提供文章、博客供大家阅读和培训。别指望人们在不经过任何培训的情况下就能掌握新实践。如果担心培训的成本，不妨测量一下修复一个缺陷的成本。（回顾上一个项目，在最终测试阶段和正式发布之后，修复一个缺陷的成本是多少？我的经验法则是：一般来说，培训的成本要低于在正式发布之后修复一个缺陷的成本，而且比修复10~20个发布前缺陷的成本也要少得多。）

接下来，寻找志愿者。不要强制推行TDD实践。可以这么说："我很担心在项目收尾时发现太多缺陷，到那时能做的事情就少多了。我听说TDD可以帮助解决这个问题。有没有人愿意试上几周，再告诉大家结果如何？"

可能一开始没有志愿者。如果真没有，那就准备度量开发速度（请参见图11-1）和缺陷（请参见图11-2）吧。记住，除非一个功能已经开发完成、达成了里程碑条件，否则它就不能记入到速度图中的"完成"部分。项目经理可以在已经设置好的大型可见图上向团队展示这些图表。

到项目结束时，特别是在缺陷数目居高不下的时候，在将修复代码签入到代码库之前，项目经理就可以着手推行TDD实践了。一旦开发人员尝试过一次之后，他们就很有可能在下一个项目中继续使用TDD。[1]

① 可以到http://www.testdriven.com和http://behavior-driven.org获得更多关于TDD的信息。

布莱恩是一个不相信单元测试的开发人员。但是项目团队决定：每个人都要编写测试。布莱恩也只好如此。如果一段代码需要四个测试，他会写四个测试。可他写的四个测试非常类似，根本无法覆盖代码的逻辑。他把一个测试写四遍，不同的仅仅是数据。（对于某些代码而言，不同的数据会有帮助，这个例子中不是。）

等到修复缺陷的时候，布莱恩有另外的花招。他写的单元测试会针对要测试的代码，而其中已经修复的部分代码就不管了。

最后，项目经理把他踢出了项目组。即使你的开发人员声称他们在做单元测试，也不要轻易相信。

如果开发人员没有做单元测试，也别去审核或是惩罚他们。要让他们认识到单元测试的价值，向大家介绍TDD，解释你为什么需要单元测试。跟踪修复缺陷的成本，并解释这些已找到的缺陷很有可能在单元测试时发现。在布莱恩的案例中，项目经理可以让大家做同事代码评审和单元测试。布莱恩也就能更早地从同事处获得反馈。项目经理要利用自己的影响力，号召大家主动进行单元测试。

13.4 使用多种测试技巧

测试可以昭示项目中的风险，还能减少技术债务（请参见附录B）。测试的范围越广泛（所测试的系统级别越高），就越能发现产品整体上的风险。如果测试的重点放在某个特定功能之上（透明程度高的白盒测试），就可以减少技术债务。

我遇到过很多项目团队，他们只做系统测试，就是让测试人员手工进行黑盒测试。以手工黑盒测试作为唯一的测试技巧，根本无法保证复杂系统的正确性。

我在提及向项目中集成测试时，是指所有的测试，而不仅仅是测试人员所做的事情。在图13-1中，你可以看到有许多种类型的测试。

13

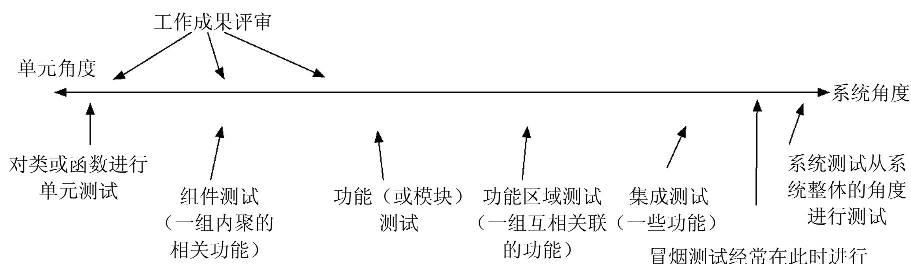


图13-1　完整项目中的测试连续体系

单元级别的测试是开发人员的工作，目的是要测试开发人员将要编写或刚写完的几行代码的正确性。根据我的经验，开发人员很难保证100%的单元测试覆盖率，除非他们先写测试。

开发人员把一些功能单元组织起来，看看能否完成预期想要的功能，这时就可以进行组件测试了。再次说明，如果使用测试优先开发的方式，组件测试就很容易进行（也能通过组件测试了解团队完成的工作）。

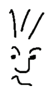
从整体上测试一个功能，这就是功能测试。如果没有采取按功能逐个开发的方式，（为什么不这么做？）这也有可能是模块测试。我总是让开发人员先进行功能测试，这样他们就能验证是否实现了本应完成的功能（正向路径）。然后，我会让测试人员检查这些功能有没有做它不应该做的事情（负向路径）。

功能区域测试的目的是要检查一系列互相关联的功能，这更像是测试人员的工作，开发人员不应负责。

集成测试会发生在多个步骤中。如果开发人员使用持续集成，那就不用再单独做正向路径的集成测试了。在测试人员开始看代码之前，你很难知道代码中是否存在难以发现的副作用。

开发人员要负责编写冒烟测试，这些测试会验证：在某个版本构建完成之后，系统确实能够做些什么。

测试人员会开发并运行系统级别的测试。



小乔爱问……

为什么我要关心单元测试覆盖率?

你可能认为100%的单元测试覆盖率（基本的路径覆盖，覆盖每个决策路径）会引起很多疑问，特别是对于从未用过TDD的开发人员来说。但是100%的单元测试覆盖率能够给开发人员和项目带来下列好处。

- 如果需要重构一段非常凌乱的代码，你就可以提前知道：测试将会告诉你是否曾经犯过错误。
- 单元测试能促成更好的接口设计，因为开发人员必须使用自己的接口来测试代码。①
- 最后，100%的单元测试覆盖率能够让最初负责的开发人员转向其他的工作，因为从测试中可以清晰了解代码的工作方式。

即使达到100%的覆盖率，代码也有可能出现逻辑错误。你很难知道代码中到底有哪些问题，但这总比没有任何测试或是只有一点点单元测试的状况好得多。

① 请查看http://haacked.com/archive/2004/12/06/1704.aspx。

工作成果评审（包括伙伴复查、同行复查、正式检查）可以在很多层面上进行。从事技术工作的同事们可以评审其他同事的工作，不管这些人是负责开发还是测试。

降低产品中未知问题造成的风险是测试的目的。为了降低风险，要确保每个团队成员明白自己在测试工作中的角色，使用多种测试技巧，并将测试尽可能地与开发同步进行。

当项目经理确定每个团队成员如何测试产品之后，就可以考虑使用测试连续体系了。开发人员从图13-1的最左边开始，完成尽可能多的测试工作（单元测试），直到集成测试。如果团队中有发布工程师或是测试人员可以编写很好的冒烟测试，也许让这些人完成冒烟测试是个不错的主意。

13.5　确定每个团队成员在测试工作中的角色

开发人员很容易限定自己的工作，认为自己的职责就是设计和编写代码。他们总是看不到测试的必要性。“如果你对代码的质量不关心，我可以开发出你需要的全部代码。只要一个下午，而且是用一只手。”但是这样的代码能正常工作吗？门儿都没有。

我敢打赌，项目经理一定想知道代码能否正常工作；而且我敢说，你的开发人员对此也同样关心。有太多的开发人员都没有意识到：他们在编写代码的同时，也在创造缺陷。减少缺陷的方法就是持续评审工作成果，这也有助于更快发现缺陷。

如果大家一直都在测试，那为什么还需要系统测试？当系统所有部分都组合在一起之后，有些问题才会显现出来，系统测试能够帮助团队发现这些问题。没有系统测试，团队要想进行其他类型的测试就得花费非常多的时间和精力。你能承担多少风险？你的团队是否准备好要发现自己所有的问题，并为此承担责任？没有多少团队能做到这一点。

管理（基本上）没有系统测试的项目

公司某个大人物把项目团队成员叫到一起，说道：“我们要在4个月后发布一个新版本，它要具备这些新功能。你们去搞定吧。”

团队成员一起开会，大略估算了工作量，怀疑要用8个月才能完成。使用现在的人力配备，他们不知道该怎么样才能在4个月内完成项目。

项目经理请求分配更多开发人员——没有更多人力了。他请求分配更多测试人员——也没有可用的测试人员了。他跟项目团队这样说：“我们必须改变工作方式，因为没有足够的人力。”

开发人员决定：他们会复查所有的代码，并为所有的代码编写自动化单元测试。而现存唯

一的测试人员负责维护产品现存的自动化回归测试。

一开始，开发人员会在完成编码后写单元测试。但是，他们偶尔会忘记编写所有测试，并在代码复查时发现了这个问题。于是，开发人员决定尝试TDD。

项目团队在5个月后发布了产品（而不是大人物承诺的4个月）。而且，他们用了很多时间编写发布后的自动化系统测试。

产品获得了某个奖项，并赚取了足够的利润，也防止了裁员。这个项目团队继续使用他们的TDD和代码复查实践。

这个团队之所以能够成功，是因为开发人员非常认同所采用的测试和复查实践。（而且产品也没有GUI界面。产品的GUI若很复杂，那这个产品就很难由开发人员独自完成开发和测试了。）开发人员若对所采纳的实践有任何懈怠，他们就不可能成功。

系统测试还是必要的。如果你的项目团队做不到像上面这个团队这样，你需要系统测试；如果你的系统有很多GUI，你需要系统测试；如果你想降低整体上的技术风险，你需要系统测试。

你的测试人员称职吗

复杂系统的测试可一点儿也不比开发轻松。对于复杂系统来说，如果创建的单元测试仅仅覆盖每条路径或是每个对象，这还是不够的。测试还需要产品专家的参与。有时，你需要理解系统设计的人参加，即使他们没有任何编码背景也没有关系。有时，你需要那些传说中的、擅长探索性测试的人，他们总是想如何才能破坏软件。有时，你需要的测试人员要可以开发可维护的自动化测试。系统规模越大，它运行的时间将会越久，这个时间可能超出任何人的想象。这时，设计良好、开发完备、不需要太多维护成本的自动化回归测试，会很有助于节省资金。

如果你认为测试人员的存在就是为了找到代码中的缺陷，那你就是没有完全发掘出测试人员的全部价值，任何新雇佣的测试人员也无法跟上开发人员的步伐，你的测试团队也就沦入二流。花些时间，按照图13-2中的问题调查一下吧。

对于图13-2中的问题，你的回答若有一半为“是”，那你的测试人员水平就是二流的。他们被排除在关键的讨论之外，无法获得工作必需的工具和产品专业知识。项目团队也许不是有意为之。大抵来看，测试人员对于测试的技术面了解并不深入，他们也不具备足够的知识和技能去测试产品。管理层害怕“浪费”资金雇佣有其他专长的人员，因为经理们觉得无法收回足够的投资。

问　题	是	否
你的测试人员是否经常被排除在需求或设计会议之外？		
你的测试人员是不是要借助偷听的方式才能了解到产品信息？		
你的测试人员对于工具的需求是不是经常被推迟或忽略？		
产品的可测试需求的建立是不是经常被推迟或忽略？		
测试人员每个人平摊的培训经费预算是不是要远少于开发人员？		
你的所有测试人员是不是可以互换的？也就是说他们掌握类似的技能，谁负责哪个项目的工作都无碍大局。		
对于非敏捷项目来说，是不是只有在产品构建完成之后，你的测试人员才能跟开发人员一起工作？原因是不是由于他们没有被尽早带入到产品的需求和设计阶段？还是因为他们对于需求和设计了解不足，因此无法提供反馈？		
对于敏捷项目来说，你的测试人员是不是只跟产品负责人一起编写测试？其原因是不是由于他们无法完全理解产品内部机制，因此无法跟开发人员一起开发更复杂的测试？		

图13-2　你的测试人员是不是二流水平？

但是，如果你真地希望节省项目的时间和资金，并且希望管理技术风险，你需要一流的测试人员。

一流的测试人员具有足够的创造力，在编码工作开始之前，他们就能评估系统的设计和架构。在代码编写阶段，一流的测试人员会设计和实现他们的测试套件，包括自动化和手动两部分，他们编写的测试会以开发人员无法预料的方式去“折磨”系统。一流测试人员会衡量已经测试过的部分，并评估这些部分的风险；他们还会思考对系统的测试是否已经到位，能否帮助项目经理弄清当前版本的产品面临哪些风险。一流测试人员会跟开发人员保持步调一致，他们会假定开发人员在使用持续集成，而且不会一次性签入一周或多于一周时间所能完成的代码。

一流测试人员与开发人员的关系是对等的。他们之间是搭档关系，而不是对手。优秀的测试人员能够改变开发人员创建产品的方式。

如果测试人员了解产品并能及早发现问题，开发人员就会创建出更好的产品，其中可供测试人员发现的缺陷也更少。为什么？因为测试人员是他们的同事，对于开发人员和其他技术人员来说，同事之间的认同感是非常重要的激励因素。

如果测试人员编写的测试可以及早发现更多问题，开发人员就拥有更多的灵活度，可以自由选择何时以及如何修复问题。要是测试人员无法在早期找到问题，开发人员就会面临困境，面对项目最后一周发现的十个必须修复的严重问题，他们要选择其中的一个或两个进行修复。不管修复哪些，开发人员都不会对自己的工作结果满意。

你可能会认为：开发人员希望以自己的产品为豪，所以他们会尽早寻找问题、解决问题。我见过很多开发人员是这样的。不过，总的来看，开发人员不是测试人员。开发人员总是无法发现自己的缺陷，所以他们很难发现自己工作成果中的所有问题。而且，系统越复杂、规模越大，开发人员就越难以发现自己的缺陷。

要是测试人员可以帮助开发人员早日发现问题，开发人员就更愿意在其他的需求和设计讨论中加入测试人员。开发人员也更愿意增加产品的可测试性，比如定义API或是其他测试可能用到的钩子方法。

测试人员要对下面这些测试有所了解或是具备使用它们的能力：边界条件测试、等价类划分、组合测试、探索测试，还有对产品的端到端测试，而不仅是按需求测试。[①]如果测试人员可以理解产品的设计或是能够阅读代码，你可以训练他们上述测试技巧。要是他们搞不清楚产品的架构和设计或是根本不懂代码，那就无法通过培训让他们获得上述技巧。两种能力，他们必须掌握其一，要是能全部搞明白，那就再好不过了。

一流测试人员不仅仅只是发现和报告缺陷，他们会将产品的信息提供给整个组织。这些信息可能包括测试结果、缺陷报告或是系统性能的数据。有时，这些信息是对于需求或设计的反馈。测试人员如果为开发人员、需求相关人员、文案人员或是其他与产品开发相关的人员提供的信息越多，其价值就越大。通过正确的方式，测试将会减少上市耗费的成本，发布的软件版本充满缺陷的可能性也就越低，还能降低进行中的维护发生的成本。

如果测试人员不是一流的，到项目后期很可能发现很多缺陷，你也得管理类似的风险。不妨使用短迭代，伴以TDD。雇佣可以帮助开发自动化系统级别测试的人，使用现有的测试人员进行探索性测试。再为下个项目进行一些相关的规划。要想完成他们的工作，你的项目团队还有很长的路要走。

① 如果想了解更多测试相关的技巧，请看［CK02］。

13.6 正确的开发人员与测试人员之比应该是多少

我在教授、指导、评估项目的时候，几乎每个人都会问："我们需要多少测试人员？"简而言之，人员数目要足以评估产品的状态。[①]

测试可以作为另一种技巧来管理风险，这些风险包括开发的产品无法达成客户需求，或是无法完成足够的测试以评估产品状态。开发人员可以（而且应该）完成一些测试，但是他们看不到自己创建的缺陷。

管理层们经常要找一些行业标准为自己的行动正名，以影响自己的上层管理者。确实有一些类似的可用资源，包括［CS98］和软件工程FAQ。[②]这些资源中提到的开发人员与测试人员之比从1:1.5到10:1不等，涵盖的组织包括微软，他们的比率是1:1［CS98］。变化范围如此之大，你不能随便选一个别人的比率拿来就用，必须要分析自己的处境，然后得出一个合适自己情况的测试人员数目。分析时要考虑下列因素。

- 需求、产品规模、产品的复杂度。产品越复杂，测试也越复杂。
- 你所在组织开发产品的方式，你的客户对于缺陷和延迟交付的忍耐程度。项目生命周期越接近顺序式生命周期，需要的测试人员也就越多，除非你的开发人员对于寻找缺陷特别积极。
- 你的开发人员和测试人员的能力和职责，还有他们被分配到项目中的时间阶段。

每个组织的上述因素都有所不同，所以你和所在组织必须对产品、流程和人进行分析，再做出选择。如果产品和流程不变，也许你可以应用同样的分析结果，做出同样的决定，并以之作为未来项目的基本准则。

13.6.1 产品风险对比率的影响

产品的复杂程度会影响测试人员的数目。复杂程度越高，需要做的测试工作也就越多。测试不一定要由测试人员完成。有些测试可以自动化，而且开发人员很容易就可以搞定。但要想知道整个系统的整体运行状况，最好由专业测试人员完成。

图13-3是对产品复杂度的一种评价方式。

13

① 请参见http://www.jrothman.com/Papers/ItDepends.html。
② 请参见http://www.faqs.org/faqs/software-eng/testing-faq/。

产品风险	系统是否存在该风险
大型系统	
复杂的、实时的	
API定义不完整	
GUI依赖数据	

图13-3　产品风险的潜在区域

也许你还会提出产品其他的风险。我见过有些系统，它们的增长超出了架构所能处理的能力范围。也就是说，要想向系统添加更多功能，同时不导致问题，这几乎是不可能的。你在评估产品的时候，可以在认为系统存在当前风险那一栏中画个X。图表中的X越多，要做的测试工作也就越多。

13.6.2　项目和流程风险对比率的影响

项目采取的生命周期越接近顺序式生命周期，你需要的测试人员也就越多。原因在于，即使项目团队一直在评审文档，但开发人员却从未收到代码相关的反馈（因为还没有构建呢）。到最后，团队需要更多的人力和时间来检查系统的整体运行过程。

在产品开发过程中，如果团队使用提供反馈的开发实践的数目越多，比如TDD、单元测试、结对编程、代码复查或正式检查，那么需要的测试人员就会相应减少。这些实践用得越少，就得配备更多测试人员，因为开发人员无法获得对于自己代码足够多的反馈。

如果开发人员更多地使用“小石子”（请参见8.10节），而且更多地按功能逐个实现（请参见9.3节），他们就对自己的工作有更深入理解。开发人员要是能更深入理解自己已经完成的和尚未完成的工作，项目需要的测试人员就可以更少。

如果使用顺序式生命周期，好几种项目和流程的风险会导致项目中需要更多测试人员。请参

见图13-4，在你不常用的技术旁边标记X。得到的X越多，需要的测试人员就越多。

项目和流程风险	如果你在项目中没有这么做，请标记×
我们总是会明确说明和管理我们的需求	
我们知道如何随项目推进评估产品架构	
我们会评估每个功能的设计	
我们按功能逐个实现	
为了进行自动化单元测试，我们拥有单元测试基础架构，而且也在使用	
我们有测试基础架构，供自动化系统测试使用	
我们在进行每日构建	
我们会在跨职能团队中评审未解决的缺陷，以评估其影响	
我们拥有并使用每日构建和冒烟测试	

图13-4 潜在的项目和流程风险

13.6.3 人员及其能力对比率的影响

开发人员能力越强，需要的测试人员越少。现有测试人员的能力越强，需要的测试人员越少。如果雇佣的开发人员不能测试自己的工作，或者雇佣的测试人员种类单一，而且他们无法深入测试产品，那你就需要更多的测试人员。

测试人员的数目不仅受一个开发人员的影响，而且他们能力的差异也是决定测试人员数目的

关键。DeMarco和Lister［DL99］在编码演习中发现了技术团队成员能力的巨大差异。

虽然一个人的工作经验能够决定工作效率的高低，但并非总是如此。有的开发人员和测试人员表现优秀，可他们并不一定资格最老、经验最丰富。与工作效率关系更为密切的，是一个人的自律精神、对相关知识的理解，以及解决方案空间域的专业知识。人们在不同能力上的确有巨大差异。

考虑到与人相关的风险时，请参见图13-5。得到的X越多，就需要越多的测试人员。

与人相关的风险	项目中存在吗
我们的开发人员对这类产品没有多少经验	
我们的测试人员对这类产品没有多少经验	
我们没有在开发人员的实践中构建反馈机制	
测试团队的测试能力有限，为开发人员提供的反馈也有限	

图13-5　与人相关风险的潜在区域

我曾参加过一些项目，其中开发人员与测试人员之比是1:1，我们没有进行足够的测试。在参加过的另外一些项目中，开发人员与测试人员之比是6:1，我们也完成了足够的测试。不存在永远正确的比率。

不过你可以向自己和项目团队提出下面这些问题。

- 现在，我该如何估算测试这个产品需要的工作量？
- 针对这个产品，我们需要哪些类型的测试？
- 要想完成这些测试工作，我们需要哪些类型的测试人员？各需要几人？
- 为了跟上这个项目开发工作的步调，我怎么知道需要多少测试人员？

复查产品的属性，计算做过的X标记，看看你的项目风险有多高。然后再看看你的团队如何开发项目。你能发现任何主动的缺陷查找活动吗？如果能，这就可以降低测试不足的风险。然后再看看项目的压力如何。

项目是否要遵循严格的交付日期？你的客户是否难以容忍交付的产品中带有缺陷？这会增加风险，也暗示着需要加入更多测试人员。

然后再看看项目团队的能力。开发人员知道如何创建低缺陷率的产品吗？他们是否深入了解产品？你的测试人员灵活度有多高？他们能否根据项目阶段、测试产品的不同改变测试方式吗？如果找不到人具备这些能力，你的风险就更大了，也因此需要更多人员。

对于开发人员与测试人员之比来说，没有放之四海而皆准的答案。在回答问题前，要先深入分析。具体问题具体分析——真正要分析的，是项目的风险以及因此而做出的权衡。

13.7 让测试与开发同步进行

测试会让项目的风险展现于众人面前，大家越早看到这些风险越好。在采纳顺序式生命周期的项目中，要让测试人员参与到需求分析阶段。询问他们关于产品需求的反馈。在采用迭代式生命周期的项目中，请测试人员帮助评估原型。使用增量式生命周期的项目，只要有可供测试的部分，就可以让测试人员尽早开始测试功能。在实施敏捷的项目中，要确保测试人员与开发人员一起工作，以开发技术层面的测试；还要让测试人员与产品负责人一起，编写面向客户层面的测试。①

13.8 为项目制定测试策略

团队中若有测试经理或是测试带头人，就要由他们负责制定测试相关的策略。项目经理要审查这些策略，并确保你同意他们对于风险的评估。

要弄清楚是否需要正式的产品系统测试阶段。如果你处在受管制的行业，而且使用敏捷生命周期，那就可以把正式的系统测试整合到每个迭代之中。项目经理要跟审核人员说明你的工作方式。在其他类型的生命周期中，项目经理需要在最后完成一些系统级别的测试，因为集成工作是在这时进行的。最后的测试阶段要多正式，只有项目经理最清楚。

13

13.9 系统测试策略模板

下面是一个系统测试策略模板：

- 产品版本和概览

① 请参见http://www.testing.com/cgi-bin/blog/2003/08/21#agile-testing-project-1。

- 产品历史
- 待测试功能
- 不需测试功能
- 包含和排除的环境配置说明
- 环境需求
- 系统测试方式
- 系统测试入口条件
- 系统测试出口条件
- 测试交付物
- 其他参考文档

产品版本和概览

描述产品和版本说明。简要描述产品的工作方式。必要时可参考其他文档。

产品历史

用三四句话，对该产品之前版本的历史进行概括说明。包括缺陷历史。（缺陷历史可以体现技术债务的程度。）

待测试功能

列出所有待测试的功能。以最有意义的方式组织该列表：通过用户功能或是通过架构领域。若能参考需求文档，不妨这么做，但不要列出所有的需求。

不需测试功能

如果知道有哪些功能不必测试，罗列于此。这一项可能只有在小版本的发布中才会用得到。

包含和排除的环境配置说明

列出该项目中将要测试和不必测试的硬件和软件配置。如果有必要，可以使用矩阵展示要测试的和被排除的选项。

环境需求

列出测试人员需要的特定测试环境。他们可能需要相对独立的网络环境，访问一台大规模的服务器，特定的固件，或是特定的软件。

系统测试方式

说明你将如何保证测试在必要的时候进行。如果打算使用与目前开发过程不同的生命周期，这里应该明确指出。如果计划使用组合测试来覆盖众多不同的配置环境，请在这里说明。此处专门用来说明系统测试相关的里程碑，可供项目团队查看。

系统测试入口条件

产品在开始正式系统测试之前，必须符合此处列出的条件。如果使用非敏捷生命周期，项目团队很容易认为自己已经达成了里程碑。然后他们准备开始系统测试，却发现产品无法运行。如果你列出了产品进入系统测试的最低入口条件，测试人员就会知道他们是否处于正式的系统测试阶段。（他们仍然可以测试，不过不是正式的系统测试。）正如2.3.3节中提到的，要确保系统测试入口条件符合SMART原则。

系统测试出口条件

产品在结束正式的系统测试之前，系统必须满足此处列出的条件。如果你打算在系统测试完成后马上发布，系统测试出口条件可以与发布条件（请参见2.3.3节）相同。（在发布前，有些组织还会要求完成额外的用户验收测试。）

测试交付物

如果要保留测试的产物，比如日志、自动化测试、计划或度量标准，可以详细列举于此。

其他参考文档

项目经理可能要在策略中引用需求或是其他规范文档。

13.10　QA 与测试有差别

我可没说要把QA集成到项目中。QA是指质量保证，这是流程改进和流程度量活动。尽管有很多测试小组都被叫做QA，但他们不是，他们是测试组。

这只是一个用词不当的问题吗？我为什么要这么关心这个呢？

流程改进是管理活动。可能会有一个QA经理负责流程度量，并管理测试。但很可能挂着QA经理头衔的人只管理测试。含义不正确的职位名称，可能会让高层管理人员觉得他们已经覆盖了相关活动——比如流程改进和流程度量，然而事实并非如此。这个职位名称在组织中意义不大。

项目经理要能够分辨测试人员与QA人员。他们的关键区别在于：QA团队人员能够改变产品

开发流程。

名副其实的QA小组或经理具备如下特质。

- 有权有钱，能够为开发人员（或技术文案、测试人员、发布工程师、业务分析师，产品开发过程中的任何人）提供必要的培训。
- 有权解决客户的投诉，或是有权推动客户投诉的处理，特别是在产品下一版本的需求优先级排序过程中。
- 有权、有能力修复缺陷。
- 有权、有能力编写或重写用户手册。
- 有能力了解客户需求，并据此设计产品。
- 有能力根据多个项目度量产品开发过程，对比结果，并解释这些结果——而且不会因此被解雇。
- 有能力了解当前产品开发过程，并有权对其进行修改。

QA经理或小组也许不会直接干这些工作，但是他们有能力安排相关的人力和工作。

QA经理和QA工程师是很重要的角色。然而，组织更愿意提供带有这些名称的职位，而不愿意为人们赋予这些名称应有的权力和责任。

测试是一种值得自豪、有创意的职业。优秀的测试人员就像优秀的开发人员一样稀有。而且，成功的项目离不开他们。不要被头衔愚弄。

铭记在心

- 如果项目经理有在项目中集成测试的规划，你就可以做到。
- 使用TDD，可以提升产品的设计和代码的质量。
- 可以考虑在项目中使用测试连续体系。

第 14 章 管理工程

你可能会对自己说："好吧，JR。我管理这些简单的项目没有问题，但是现在我不再负责简单项目了。我要管理的几个项目会在同时发布产品；或者，出于战略决策考虑，我要管理一系列项目。要同时管理这么多东西可真是一场噩梦！你有什么好建议吗？"

没问题。你所描述的可被称为工程，而你要做的工作是：工程管理。

14.1 如果项目是工程

下面是对工程管理的一个实用定义。[①]

工程管理：协调多个子项目或是一系列项目，以达成特定的业务目标。

项目管理是战术上的。项目经理的职责是完成自己手上的项目，而不会考虑其他进行中的项目。但是工程经理的职责更注重战略性，因为工程经理要协调多个子项目或是一系列项目。工程经理必须时刻将工程的战略业务目标牢记在心，而且要保证每个项目都遵循这个目标。如果目标发生变化，工程经理还得（与相关项目经理）决定各个项目要做哪些调整。项目经理会完成具体的调整措施。

有这样一个例子。我们假定你被任命为一个新在线业务的工程经理。人们可以在你的站点上购买和销售有形的产品和软件。也许你会发现，整个大项目可以很容易切分成多个子项目，比如购买有形产品、购买软件、销售有形产品、销售软件。而且，为了向买方和卖方推广这个业务产品，你可能还需要同时准备营销文件和销售计划。所有这些项目都是一个工程的组成部分，而且有同样的目标：该站点成功上线。

假定各个子项目都在取得进展。现在已经进入了整个工程开发的中段，买方子项目已经可以投入使用了，但是卖方子项目的税务计算功能还有点问题。你的律师很难判断出哪些缴税人比较

① © 2007 R. 麦克斯·怀德曼，http://www.maxwideman.com，经许可后复制。

可靠，也不知道他们缴税的地点在哪个州或是哪个国家。而且，购买软件这个项目中，有些具体场景的性能还不能令人满意。

工程经理要做决策（与公司管理层一起），判断是否应该在律师搞清楚税务事宜之前暂时停止整个工程，或是搞清楚是否可以先发布站点的购买功能，稍后再加入销售功能。如果你是负责购买部分功能的项目经理，就不会关心工程经理的决策，除非这些决策影响到你的项目，改变你的项目与整个工程的整合方式。

工程经理在进行战略管理决策时，要以各个项目之间的依赖关系作依据。这些依赖关系有时体现为可交付物的形式。更多时候，这些依赖关系涉及如何在项目组合中调配人员和资金。要管理人员和资金的分配，请参见第16章。

14.2　将多个相关项目组织到一个发布版本中

管理工程与管理项目类似，但是范围更广。特别是在试图整合跨组织（不仅仅是跨技术组织）的人和可交付物的时候，工程经理要具备影响力和谈判技能。不过最先需要的还是规划。

很多时候，我使用类似于下面工程规划的模板。我总是确保整个工程以可交付物为重，同时让人们注意如何保证自己的可交付物不出问题。

下面是一个工程规划模板：

- 综述
- 功能
- 工程需求
- 工程目标
- 市场评估和推广计划
- 销售计划
- 投资回报率和产品生命周期
- 日程综述
- 人员安排曲线
- 培训计划
- 服务/支持计划
- 其他计划

综述

提供产品的简要概述。为什么要开发这个产品？该产品要在公司的业务规划中达到什么目

的？

功能

说明工程的重要功能——高层产品需求和目标。包括国际化的问题。如果你在处理一个产品的多种模型，要说明各种模型之间的差别，或者指明人们从哪里可以了解这些差别。

工程需求

就像一个项目一样，工程也会有自己的需求和目标。

工程目标

工程目标与企业的目标类似。

市场评估和推广计划

包括预估的产品生命周期、整个生命周期中的收益曲线、估算支持成本，以及什么都不做的成本。

销售计划

回答如下问题：期望销售周期、期望销售路线，诸如此类。

投资回报率和产品生命周期

包括预估工程成本、预估在没有该工程时会发生的当前和未来成本、预估将来工程完成之后会发生的成本，以及预估的投资回报率。

日程综述

设计日程中主要的里程碑。如果有演示或商展的日程要标注出来，我会将达成这些日期标注为风险。

人员安排曲线

如果你在争夺整个组织中稀缺的人力和资本资源，要在此处说明这些需求。要估算出需要的人员数目。

培训计划

在此处说明是否需要对内部或外部人员进行培训。

服务/支持计划

如果面临服务或支持的问题，要在此处说明。

其他计划

工程可能还需要运营计划、硬件发布计划、部署计划，也许还有别的。要把这些议题加入到工程计划中。

在制定工程规划的过程中，工程经理可以识别风险。你会发现，如果有独立的工程风险列表，就更易于管理风险。我经常发现，我的第一版工程规划大概有5页，初始的风险列表会有好几页。随着工程各种状况更加清晰，规划会变得更长，而工程风险列表却会缩短。有些风险转移到了项目之中，有些可以在工程和子项目中发现。

14.3　随时间推移组织多个相关项目

随着时间推移，在多个相关的项目中，要想跟踪哪些需求随哪个版本发布会变得非常困难。使用敏捷生命周期可以让你的管理变得异常简单，因为产品在任何一个迭代结束时都有可能拿去发布。接下来要做的，就是决定要在什么时候发布哪些功能了。

如果无法使用敏捷生命周期，尝试使用“发布列车”吧，这对实际的版本发布会有帮助，同时还能有效管理产品未来版本持续不断变化的待办需求列表。

好吧，我解释一下。“发布列车”就是为期三个月的迭代。这个待办事项列表是敏捷生命周期所使用的，不过敏捷生命周期会使用更短小的片段，因为客户或其代理人希望一直见到产品。实际上我想让你使用敏捷实践，差别在于“发布列车”的风险更高，因为时间盒的长度更长。

14.3.1　将产品的多个版本组织到发布列车中

发布列车[1]是按季度组织功能的方式，目的是要定期发布产品的多个版本。

如果你是项目经理，而且知道自己要定期发布产品，就可以使用发布列车。如果你是工程经理，也可以使用发布列车来管理产品发布策略。

发布列车将发布版本与项目分离开来。发布列车是要按季度交付产品，通常在一个季度同一个月的同一天（比如第二个月的第十天）。已完成的项目，即使仅仅包含一个小小的功能，也能

① 请查看http://www.jrothman.com/weblog/2003/08/release-trains-help-manage-resources.html。

放到列车中交付出去。未完成的项目不会交付。将工作切分成更小的项目，可以确保发布列车正常运作。

如果你遇到了诸如6.11节中提到的日程游戏“我们必须拥有这个功能，否则就完蛋了”，发布列车可以帮你摆脱困境。发布列车只会改变给定项目的一个维度。你不必将一大堆功能都塞到某个版本中，而是可以组织多个并行的项目，逐步实现所有功能。

有些项目经理使用发布列车去分离“增加新功能”和“改进现有功能”的工作。你也可以将功能和性能相关的工作分成不同的小块工作，再为每一小块工作安排在哪个版本发布。可以同时开始多个版本的相关工作，这样一来，你仍在开发同样的产品，只不过在不同时间的交付物不同。

使用发布列车，工程经理可以安排很多独立的功能。发布列车让你可以同时管理多个增量项目。只要有功能完成，就能把它们加入到列车中，再发布出去。进度快的功能不会受进度慢的功能的影响，仍然可以成为产品的一部分，或是交付出去。即使工程经理要改变功能的发布顺序，你也有更大的灵活性。

14.3.2 让发布列车为你服务

使用发布列车并不难，但是对项目和工程有如下一些适用条件要求。

- 工程经理必须与制定项目组合的人一起工作，请参见16.1节。为了保证产品每个版本发布都不出问题，工程经理要管理产品的整体发布条件、各个项目经理，还有其他与公司相关的议题。工程经理还要决定某个功能是否要错过某次列车，以及它是否应该推迟到下次列车中。任何没赶上列车的东西都要返回到产品待办事项列表中（请参见16.6.1节）。工程经理要帮助各个项目保持其内聚性，并减小项目之间的耦合性。
- 项目经理管理各个项目，工程经理管理列车。每个项目都包含很多工作量，至少需要一个技术带头人，还得有个项目经理，以确保开发人员和测试人员（以及其他必要的人员）能够估算并按时交付工作。项目经理必须确保开发人员不会完成非必要的功能。YAGNI原则（即you aren’t going to need it——你不会需要这个功能）不仅仅适用于敏捷项目，对所有的项目都有效［JAH02］。项目经理要具备必要的技术能力，以理解项目中必然会发生的权衡过程，还有对于加入和去除哪些功能的讨论。
- 工程经理需要很好的软件配置管理过程，既包括工具也包括发布工程师如何使用这些工具。可能需要同时提供四条代码线（在一年之中，每个发布列车使用一条），而开发人员仍需构建他们负责的源代码。发布工程师必须将不同代码线合并到主线中，这样就可以在交付下个版本前发布必要的补丁程序。（要尽量避免发生这种情况，除非公司会因为不发布该补丁而破产。）

- 每个人（开发人员、测试人员、项目经理和其他项目相关人员）都要具备良好的估算技能，工程经理才能按时发布每个列车。
- 团队要有足够的自动化测试，供回归测试使用。发布列车就是将迭代式生命周期与每个发布版本的增量结合在一起。工程经理必须保证同样的模块可以反复修改。如果你没有足够的自动化测试用例（对不起，我无法提供一个固定的数字），你就无法按时发布。

为了创建成功的发布列车，工程经理必须要具备上述这些条件。

使用发布列车，工程经理就可以管理所有的资源，因为你可以将类似规模或类似作用的项目进行归类，并放到一个工程中。由于工程经理的职责就是让管理层和其他企业的股权持有人满意，因此一个项目无法全面“胜过”另一个项目。在项目层面发生的局部优化不能在工程中发生，要不然工程就无法准时发出发布列车了。

14.3.3　在不使用发布列车的情况下，将多个版本组织到一个产品中

如果你不想使用敏捷生命周期或是发布列车，就会发现很难同时管理多个版本。我不知道如何能让顺序式生命周期或迭代式生命周期正常运作，而且仍能完成期望的发布日期要求。如果工程经理不必面对发布日期要求，也许你可以使用其他生命周期模型完成目标。不过，除非用了某种增量式或敏捷生命周期，否则发布的顺序很快就会支离破碎，导致最后不能发布任何东西的风险会很高。工程经理无法得到足够的反馈去改进各个项目，更不用说整个工程了。

14.4　管理项目经理

无论你管理的工程是要将多个子项目的产品集成为一个版本发布，还是要随时间进展发布多个版本，工程经理都要管理项目经理。

在管理项目经理时，要确保以可交付物为准绳，还要考虑管理异常情况。也就是说，你要假定人们尽职尽责，而且是在应对问题，除非他们明说自己做不到。工程经理要跟项目经理之间建立信任关系，还得让自己明白他们的工作状况。想知道更多细节，请参见［RD05］。

当工程经理在管理异常情况时，总是会发生一个问题：存在过于乐观的项目经理，或是对项目度量不够的项目经理。你需要跟他们一起审查他们的项目仪表板，确认项目在向前推进；审查他们的风险列表，看看他们是不是在管理自己的风险。

要鼓励你的项目经理们使用敏捷或增量式生命周期，每个人可以因此得到频繁的反馈。不要依赖预测，也别让你的项目经理这么做。要依赖反馈。

如果工程经理陷入到顺序式生命周期之中，而且你还是不愿意跟项目经理一起使用时间盒或是增量式交付方式，或是客观上不能这么做，那我只能祝你好运了。只有如此你才有可能成功。

从项目经理处了解情况

工程经理更应该避免召开顺序式工程进度报告会议。你会面临成堆要解决的问题，不要浪费时间去获取每个人的进度。另外，由于工程经理要跨越组织把很多交付物整合在一起，你的管理层可能想看到甘特图。所以，工程经理需要一个工程和项目协调人，专门负责管理甘特图。工程经理仍可以使用黄色即时贴开始规划项目（我推荐这么做），不过你还得跟希望看到甘特图的人报告进度。你需要有人专门使用日程安排工具，而你要完成管理工程这样的超难工作。

提示：用即时贴管理跨工程的日程变更

如果你需要变更工程的日程，使用即时贴吧，这样每个人都能看到谁在何时、做什么样的事情。你也许可以使用甘特图检验工程的进度。但是仅使用甘特图来变更日程是不够的，要确保日程以大型可见图的方式展示。通过那样的方式管理。

工程经理可以让项目经理要求团队成员用小石子安排任务。你的职责是与项目经理一起制定交接物的日程，并让每个人都看到相互工作的依赖关系。项目经理管理团队成员如何达成他们的可交付物。工程经理的职责是要寻找障碍并移除障碍。

要让项目经理每周给你发送进度报告，说明他们的进度离交付还差多少。如果项目经理们都使用敏捷生命周期，工程经理看到的就是他们的速度图（图11-1）和迭代内容图（图11-2）。

如果有些项目经理使用增量式生命周期，那么他们在开始开发和测试之前就无法提供这些图表数据。要让他们使用时间盒，以限定项目中编码之前的工作，这样工程经理就可以尽早见到交付的价值。

与项目经理一起每周审查一次他们的风险列表，或是他们要求你这么做的时候。监控他们的风险列表，工程经理就可以获取需要的信息，移除组织内的障碍，管理异常情况，而你也不必巨细无靡、事必躬亲。这样你也可以为自己的风险列表获得需要的信息。

开始收集进度之后，工程经理就可以创建工程仪表板了。

14.5　创建工程仪表板

在管理工程时，你会希望看到所有来自项目的数据。也许你会想将一些数据放到一个工程仪表板中。

14.5.1　度量互相依赖的项目

工程经理需要各个项目仪表板（请参见第11章）的度量数据，因为你要集聚组织中许多人的努力，创建一个可交付物。也许你得集成多个子项目的度量数据，创建出一个可用的工程仪表板。有这样一个工程经理，他使用类似于故事版的机制来描述工程的进度。

描述工程进度的故事版

——艾瑞克，资深工程经理

我在管理一个很大的工程。我们有7个子项目，其中3个项目要交付硬件，并要集成到最终产品中。为了展示进度，我决定画出一个故事版，这也许是个不错的方法。

我霸占了走廊的一面墙。项目团队分散在整个建筑物中，所以我想故事版放在哪里都没太大关系，只要它们固定在一个地方就可以了。

项目开始时，我们画了十幅图，虽然这些图并不漂亮，但是它们可以展示出我们在每个迭代结束时想要的产品状态。（我们使用有时间盒限制的迭代。）我们将这些图挂在墙上，并标以“规划进度”。

每个迭代结束时，不管产品是什么状态，我们会给它照相。对不了解其内在机制的人们而言，有些照片不太清楚，所以我们会加上一些说明。我们在这些照片上标明“实际进度”，并把它们挂在规划图的上方。这些图片的下面，是每个子项目的速度图和迭代内容图。如果你看最上面一行，就会知道我们的实际工作进展。你也能看到我们最初规划的每个迭代结束后的工作进度，在最下方可以看到图表。

完成了第三个迭代后，我们认识到：当初低估了集成某些硬件需要的时间。于是我们就重新估算，然后在第四个迭代开始时写上这样一个即时贴：“搞清楚了集成问题。”有不少经理问我这是怎么回事，只要我一解释，他们就明白了。由于我们所做的大部分项目都有类似情况，所以他们对于集成上出现问题并不奇怪。不过这么早就能了解到这些问题，他们还是满惊讶的。

我们完成第八个迭代后，市场部打算从下个迭代中撤出几个功能，这让我们非常吃惊。由于他们已经看到了产品的演化过程，而且了解了每个迭代进度，因而他们发现这些功能毫不重要。

我们没有在十个迭代内完成工作。第十个迭代没有完成的工作，我们用了一个更短的迭代就完成了，第十一个迭代早早结束。

使用图片展示进度，整个组织都能了解到我们的工作进展。

14.5.2　度量一系列项目

除了使用第11章中提到的度量方法，工程经理还应该使用回顾（请参见15.4.3节），这样可以

了解到将来还要衡量哪些东西。举例来说，可以持续度量投资回报率，还可以度量修复一个缺陷的成本，或是发布一个版本的成本。如果项目总是难以按日程进行，先看看这些项目在使用哪种生命周期，然后考虑使用日程图（请参见图11-6）。要是项目使用敏捷生命周期，那就好好检查一下这些项目的速度图吧（请参见图11-1）。

铭记在心

- 工程管理需要从战略的角度看待产品，不能仅从战术角度去看。
- 工程经理要确保自己能够明确看到所有项目的进度。
- 要想清楚哪些度量方法适用于你的工程。

第 15 章

结 束 项 目

15

要结束项目，项目经理就得执行好几种活动：要求发行早期版本、主导beta测试、指导项目走向真正的尾声。不管采取哪些方式结束项目，都不要忘了做项目回顾。

15.1　管理发布早期版本的请求

项目经理如果一直在使用敏捷生命周期（而且一直随着项目进展进行测试），就不用担心发布早期版本的请求，因为软件在每个迭代结束时都是可以发布的。如果提供的功能不够，可能客户不愿意付钱，不过产品总是可以发布的。

要是使用其他生命周期，项目经理就得尽量早知道是否需要发布早期版本。回头看看你最近管理的项目，如果在大型组织中，也可以看看其他人最近的项目。这些项目要求发布早期版本了吗？如果答案是肯定的，很可能你也得这么做。

为了管理早期版本，有一些欺骗行为也没有关系。没错，你应该使用顺序式生命周期。不过，即使在编码阶段，也可以按功能逐个实现、实施持续集成，并且开发完一个功能就马上进行测试。没有人说你不能这么做。即使你把顺序式生命周期变成了按阶段交付的生命周期，这又有什么关系？也许只有你知道这样做的影响。不过在我评估过的组织中，总是能听到一个响亮的答案："没关系！"

要是开发人员做不到按功能逐个实现，欺骗行为就不会起作用，项目经理还是可以让开发人员使用持续集成，让测试人员按功能逐个开始测试。

如果这些方案都不适用，项目经理就得准备两次结束方案了。第一次是发布早期版本，第二次才是实际的发布版本。这样做的成本很高。要想避免类似情况，项目经理就要跟团队沟通，告诉他们：如果你必须这么做的话，他们会发疯的。（人们都不喜欢"穿越沙漠综合症"，请参见附录B。）

15.2　管理 beta 版本

项目经理要搞清楚关于beta版本的几个问题：希望发布几个版本、对产品完成度的要求、哪些客户将会使用beta版本？当然，这些问题的回答都基于beta版本的持续时间和目的。

可以试着将发布beta版本作为一个子项目。如果使用敏捷生命周期，在版本计划中要预估从哪个迭代开始发布beta版本。有了更多信息之后，项目经理还要及时更新版本计划。要是使用其他生命周期，要预估何时开始beta版本相关工作，再随着项目进展更新预估。

下面是我的beta测试模板：

- beta测试目的
- beta测试客户选择
- beta测试入口条件
- beta测试出口条件
- 总体beta测试日程

beta测试目的

简要描述产品版本，为什么要进行beta测试，给公司带来哪些好处，诸如此类。

beta测试客户选择

包括如何选择beta客户、初始客户名单、客户文书工作负责人等信息。

beta测试入口条件

这是一个里程碑条件，表明项目经理知道已经准备好开始beta测试。类似于发布条件，或是系统测试入口／出口条件。

beta测试出口条件

这也是一个里程碑条件，表明项目经理知道已经准备好结束beta测试。也就是说，你要说明怎么样才能知道自己已经到达了beta测试阶段的尾声。如果需要客户提供参考信息，向客户索要参考信息的活动也要放进beta日程之中。

总体beta测试日程

说明谁是beta测试协调人，或者每周选一个人负责。说明谁将负责回答beta测试客户的电话和邮件。下面是一个总体日程实例。

周　数	主要任务
第1周	与客户验证系统安装过程
第2周	确认客户已运行功能3和功能4。询问性能情况
第3周	开始索要参考信息

15.3 项目经理何时知道无法按时发布项目

在项目快要结束时的某个时刻，你知道尽管已竭尽全力，但还是无法按照期望的日程达成发布条件、交付项目产品。此时，应该找出未完成的工作还有哪些，再重新规划剩余的项目工作。

首先，要验证项目的发布条件仍然有效。你能少开发几个功能吗？（请参见5.3节）是不是还得开发更多功能？记住：发布条件是保证产品对客户有用的最少条件。

接下来，要弄清楚：需要花多少时间，才能开发完产品必备功能的最小集合。如果团队以前没有按功能逐个开发、集成和测试，赶紧开始这么做吧。项目经理如果能帮团队逐个功能进行开发、集成和测试，弥补进度所花的时间就越少。

15.3.1 “避免小的偏差”

（标题来自彼得·法格，弗雷德·布鲁克斯也曾引用过。）

项目经理要知道团队落后了多少进度。如果项目是按功能逐个实现的，而且测量过开发速度，要搞清楚人们无法完成工作的障碍是什么。很可能他们在同时处理多个任务，或是被来自其他项目的问题所干扰。如果项目延迟了，项目经理要跟团队和组织的其他人一起，确保团队的注意力放在当前项目上。项目经理和团队就可以很快地做出新的日程安排。

如果团队没有按功能逐个实现，项目经理就无法获知团队的开发速度，也搞不清楚还剩多少工作量。这时，你要将项目发布日期后延足够的时间，使得团队能够完成他们的工作。

无论如何，都要避免小的偏差。小偏差会导致“穿越沙漠综合症”（请参见附录B）。如果你曾遇到过每星期都要落下一周工作量的项目，就会知道那有多么糟糕。人们需要积极的时间表，而不是不可能做得到的日程。

15.3.2 承诺一个新日期

项目经理如果很清楚团队无法按原定日程交付项目，就要重新规划。要避免任何小的偏差，因为你知道那样做的灾难性后果。新日程要如何制定，才能让项目团队能够承诺按时交付？

“穿越沙漠综合症”会置项目于死地

每个人都投入了大量时间，团队的气氛空前紧张。你都不知道弗雷德里克上次是什么时候露面的。但是你还是完成了beta测试。接下来要做的就是完成项目。

当人们将注意力放在中期里程碑上时，他们不知道项目的方向到底是什么。他们的工作时间太长了，会将无法准时完成的项目变成死亡之旅。

当你认识到项目无法达成期望日期后，不管这个日期是中期里程碑还是发布日期，都应该重新规划、重新安排日程。你可能要从功能列表中去掉几项。你可能要放弃一些可靠性或是性能目标的要求。而且你还可能决定略过某些中期或某些版本的发布日期。

你的选择，要基于项目的成功标准和客户的要求。成功标准和发布条件能够帮助你做出正确的决策。管理层可能不喜欢你的决策，但这可能是你所能做出的最好选择了。

不管你怎么做，都不要允许项目团队只把注意力放在中间里程碑上。他们会这么做。而且你还要应对“穿越沙漠综合症”，这一点都不好玩儿。

首先要搞清楚是否要承诺一个新的日期。项目团队是不是已经实现了一些最有价值的功能？项目经理能否宣布胜利，并完成这个项目？如果可能的话，就这么做。

要是必须完成团队已经着手开发的那些功能，你就得重新规划了。

每星期落下一周的工作量是很糟糕的做法

在职业生涯早期，我曾参加过这样一个项目。这个项目每周都会落下一个星期的工作量，这样延续了4周。4周之后，整个团队都疲惫不堪，我们不知道这样的糟糕状况何时才能结束。

项目经理泄气了，说道：“我搞不清楚怎么才能知道咱们啥时候完成项目。我们为那个设备驱动程序所做的每一件事情都没有成效。我又怎能知道何时才能完成呢？”

我们慢慢陷入了“90%完成状态”的日程游戏（请参见6.14节）。由于快要接近发布结束期，现在的压力非常非常大。

我们决定改变各自负责的工作。要以全新的视角来看待问题。当我们重新分配任务并进行了简单的Delphi估算（请参见5.1.2节）之后，我们制定出了新的日程安排。当时认为还要用3周才能完成，我们决定对外承诺6周的时间。

4周之后，我们完成了工作。即使从全新的视角来看问题，有些问题还是很难解决。如果我们维持之前3周的估算，我们又会陷入士气低落、不知项目路在何方的境地。

召开重新规划会议，并邀请项目团队的所有人参会。（如果你负责一个工程，让互相独立的小组各自完成他们的规划，并呈交他们的日程。如果各组之间互相依赖，就得把大家都邀请到。）让大家使用即时贴安排日程（请参见4.3.1节），产生他们的任务列表，并且尽可能使用“小石子”估算任务。（请参见5.8节）。

弄清楚有哪些任务后，就可以开始估算了。如果项目经理希望人们承诺一个确定的日期，就得把任务的规模（任务有多大）和持续时间（任务要用多久完成）分开，请参见5.1.8节。

人们会抗拒估计某些任务的大小。“那个驱动程序我们已经开发了好几周了。我们也不知道哪里出了问题，没法给出一个日期。”要是能够对试图解决的问题有所了解，这会很有帮助。项目经理不妨提出一些问题，搞清楚某个特定的任务是已经完成了，还是陷入困境。如果项目一直是按功能逐个开发的，你也许可以借鉴一些这个项目的历史数据，用于估算。

某些时候，你会获得关于持续时间的估算。把这些时间加起来，估计一下项目大概会在什么时间结束。在估算时要多动点儿脑子。是不是有人同时还在完成其他项目的任务？增加估算时间吧。是不是现在还有人从未收到过关于他们估算的反馈，所以你也不确定他们现在的估算准确度有多高？要解释清楚：“汤姆，你刚刚得到关于自己估算的反馈，从项目开始到现在，你一直在提升。现在你愿意把自己的薪水押在这个日期上吗？”（请参见5.1.3节。）别真让人们押上自己的薪水，倒是可以用其来判断团队对新估算的把握有多大。

把持续时间加起来之后，要确保时间之和不超过几个星期，否则你的风险就会很高。（毕竟，是当初的估算把你们带到如今的境地，不要再轻易相信那些估算了，否则你们很难逃出生天。）

到了项目尾声，人们希望它能结束。而且他们希望让任何头衔中有“经理”的人知道：这个项目马上就要结束了。你不是当真要拿汤姆的薪水打赌，而是要让他认真考虑后果。项目经理可以因此而得到一个更好的估算，因为团队达成该估算的可能性更大。到了项目收尾时，如果你觉得还是要推延，这个更保守的估算可以让团队正常完成工作，而不是去匆忙赶工，增加技术债务（请参见附录B）。

15.3.3　估算系统测试时间

最终的系统测试包含如下步骤：测试产品、找到缺陷、验证缺陷。在估算最终系统测试持续时间时，项目经理首先要做的就是将这些任务分开。

如果项目经理实行测试驱动开发，并将系统测试集成到各个迭代中，最终的测试很可能都不必占用一整个迭代的时间。如果你还是个敏捷开发的新手，可能需要在结束时规划出一个迭代的时间，用来进行最终的系统测试和客户验收测试。随着团队获得更多经验，知道如何在迭代中完成功能，他们就能做更好的规划。

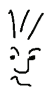
要是没有实施测试驱动开发，而且测试人员在GUI确定之前也无法着手寻找问题，项目经理就得在项目结束时空出更多的系统测试时间，而且要比预先想好的时间更长。这个时间非常难以预测。[①]

项目的生命周期越类似于顺序式生命周期，需要的最终测试时间就越长。如果类似于增量式生命周期，那最终测试时间就要不了那么多，但是需要进行更多的中期回归测试。这里的增量是指在开发过程中采用持续集成、TDD和测试。

可能有一个问题你想知道答案：一个“彻底”的测试周期要用多久？测试是不可能做到绝对“彻底”的，所以项目经理对“彻底”的定义可能是这样：运行了所有规划好要做的测试、其他探索性测试以及其他任何必须要在一个测试周期中运行的测试，这都是为了提供待测产品的足够信息。我发现这个定义还是比较模糊，而且要视各个项目情况而定。我也不知道怎样说得更明确，因为这个预估要看项目实际情况。如果开发人员在开发过程中实施单元测试和集成测试，从而致力于减少缺陷，这个时间周期可能会不断递减——因为测试人员知道如何更好地设置测试，而且产品缺陷也更少，测试也能更快进行。

项目经理知道了测试周期的大概时间之后，就可以估算开发人员修复问题的时间了。可以采纳如下数据：上个项目中每个测试周期发现的缺陷数、你对当前项目每个测试周期应发现缺陷的情况预估，发布前修复每个缺陷所耗成本数据。我曾参加过一个项目，在上个版本的第一个测试周期中，我们发现了200个问题。开发人员修复一个缺陷大概要用半个人日，我们共有10个开发人员。在第一个测试周期结束时，我们估计修复问题要用去十个人日。这个时间可不短，而且令人郁闷，我们觉得永远都不可能把所有的问题解决掉了。（提示：收集类似数据，并在回顾中使用这些数据，人们也许会因此而在后续项目中选择其他实践。请参见15.4.3节。）

现在，你已经有了对于测试周期持续时间的估算，并对修复这个周期发现的问题所需时间有了初步想法。接下来要做的，就是判断需要多少个测试周期。



小乔爱问……

在测试人员测试的时候，开发人员不能修复问题吗？

他们可以，而且我也希望他们这么做。不过，如果在你管理的项目中，开发人员不进行持续集成、不使用TDD、不复查彼此的工作成果，他们修复缺陷的速度就很难如你所愿。只有项目经理和团队知道自己的开发和测试该如何同时进行。

① 请参见http://www.stickyminds.com/s.asp?F=S8919_COL_2。

每个产品和发布的版本都是独一无二的，而且有其特定测试周期数目。开发人员寻找缺陷的积极性越高，工作方式越体现增量特征，项目经理需要的测试周期就越少。如果开发人员没什么积极性，而且工作方式接近于顺序式生命周期，项目经理需要的测试周期就越多。我听过的测试周期数字从3、8到30不等。

说起测试周期，特别是与那些希望测试在3个月前就完成测试的人们交谈，需要提供每个周期的花费时间，还要说明对所需周期数的预估。“每个测试周期要用去6天。要想修复问题，我估计在两个周期之间要用去3天。也就是说每个测试–修复周期要用去9天。现在，我认为我们需要4个周期，共计36天，也就是两个月的大部分时间，还要假定每个人都继续为这个项目工作。我们还得随着时间推进重新评估。”

15.3.4　在时间不够的情况下管理系统测试

项目经理已经尽量争取更多测试时间了，但是出资人还是说“不”。这时你该怎么办？

可以考虑用时间盒来管理时间限制。假定你规划了3个月的测试时间，可是管理层说他们希望在6星期内发布。可以用下面这些方法与测试人员、测试经理一起工作，用时间盒约束测试。

(1) 复查最初的测试计划。测试人员当时对于测试哪些功能、如何测试有些想法。要清楚地告诉管理层和其他利益相关者，测试人员无法完成最初计划中所有的测试，然后再决定你能完成哪些工作。

(2) 在第一周内明确说明测试计划的具体活动。明确说明你要如何发现应该测试产品的哪些功能，以及你将如何进行测试。为了发现哪些功能需要测试，你可以选择探索性测试；为了知道如何进行测试，你可能要了解组合测试技巧。到这周结束时，你会有一个列表，其中说明了要测试产品哪些功能以及如何测试，而且按优先级排序。可以考虑按价值高低进行测试，也就是说，首先测试产品最有价值的部分，而不一定是风险最高的部分。请参见9.3.1节。

明确说明具体计划，部分原因是为了解释用时间盒约束测试的三个主要风险：首先，你可能在测试时发现某些重要的问题，而开发人员届时没有时间修复；其次，你可能会漏掉某些重要的问题，而客户认为其非常严重；最后，你可能遇到某个阻塞性的缺陷，使得某个特定区域的功能无法得到测试。每个人都要明白：测试团队无法了解产品的一切，而组织发布的产品也很有可能跟大家最初期望的不同。

(3) 在接下来的三周时间内，要开发测试用例，并持续修改测试计划。根据优先级列表，为特定的工作流程（或是产品的某个领域）开发测试用例并进行测试。你不能白白坐在那里等着测试什么东西，而是要从头到尾测试某个工作流程或是产品的一部分；然后再测试新的工作流程或是产品领域。举例来说，如果你在测试银行系统，可以从测试建立某种特定类型的银行账户开始，再验证账户确实存在于数据库中，而且是激活账户。不能仅仅测试建立不同类型的账户，而是从

头到尾测试某特定类型账户的相关工作流程。换句话说，如果你要测试某种生物医学设备，就得测试它可以接受某个特定的输入，进行一些运算，再产生期望的输出——只测试一种特定的输入。再强调一遍，你不必测试所有的输入和输出，而是按顺序从头到尾测试各种流程。（这就是按功能逐个测试。）

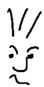
随着测试计划的不断修正，测试的范围也在不断变化。当你每次认识到有些东西无法测试时，应该把它加入到“无法测试”目录列表中，并说明对其不进行测试的风险。所有的测试完成后，使用已完成的测试计划和测试用例，你可以更新测试报告。

(4) 在每周测试工作结束时，要评估当前进度，并报告测试数据。（可以将其看作迭代结束时的回顾会议。）如果能够在测试时验证修复的正确性，那就计划在第5周继续测试，同时验证修复措施。要是做不到，那就计划下一周继续，尽可能完成第4周的测试工作。

(5) 现在到了第5周。如果你还没有验证过修复的正确性，现在应该这么做了。在动手验证时，要运行所有已创建好的回归测试，确保修复没有造成新的问题。如果这要花费你两周的时间，那就正好用完所有的时间，完成了规划的工作。要是还有一周时间，你可以继续向更多功能发起攻击，并采用之前用过的从头到尾的测试方式。

(6) 在第6周内，你要验证最后的修复措施，还要报告进度，说明你现在对产品了解了哪些，不知道哪些。

当然，我不是说你只能用6周的时间来完成一个项目的测试。测试所需的时间要由项目的具体内容和产品质量决定。不过，如果你曾经遇到麻烦，没有足够时间测试所有的东西，使用时间盒可以帮你评估只能完成哪些测试工作，从而继续为组织交付有价值的成果。



小乔爱问……

为什么系统测试要用这么长时间？

在项目结尾时，系统测试用时很长，原因在于：项目工作在取得进展的同时，测试却没有跟上，没有集成到项目中。请查看第13章。没能做到早点儿在项目中集成测试，也许你有不错的理由，但是到了项目临近结束时，你就得规划出更多的时间用来测试。

15.4　指导项目走向完成

假如一切顺利，项目经理现在所要做的就是结束项目。

15.4.1　管理“结束游戏”

看起来，项目将会准时完成（或是接近准时）。测试人员能够跟上开发人员的节奏。项目经

理也在追踪发布条件，看起来，你可以在期望的发布日期前达成所有的发布条件。那是不是现在就无事可做了？

应该继续收集缺陷相关的数据，还要确保你不会玩“缺陷提升和降级的游戏”。如果为了满足项目日期要求，项目经理打算接受更多技术债务的存在（见附录B），也没关系。不过要保证这是一个深思熟虑的决定。

如果项目经理一直在牢牢掌控项目，你接下来的任务就是规划回顾，然后就可以庆祝了。

15.4.2　避免“缺陷提升和降级的游戏”

每天早上9点，产品经理蒂姆、软件项目经理丹和工程经理苏会聚到一间屋里，进行“缺陷分类”。他们的工作是评估每个缺陷，并确保当前版本中修复必要的缺陷。

丹把目前的已知待解决缺陷列表和缺陷趋势图表递给大家，开始说道：“我们昨天又发现10个缺陷，只修复了3个。现在我们有547个待解决的缺陷，离项目截止时间还有两周。我们的状况不大好啊。”

苏说道：“咱们一起研究研究。”她和蒂姆把缺陷都过了一遍，然后开始重点讨论高优先级的缺陷，“这些缺陷的优先级其实并不高。我们可以先等等，在下个月发布的小版本中再修复它们。这些严重程度标为‘高’的，其实也没那么严重。把它们都改成中等吧。”苏和蒂姆协商了几分钟后，说道：“所有这些优先级和严重程度中等的缺陷，我们可以把它们改成‘低’。现在就只有一个严重程度为‘高’的待解决缺陷了。你可以在下周结束前修复它，没问题吧？”

丹坐在那里，愣了一小会儿。他缓过劲儿来后，说道：“可能我没太搞清楚。我们找到了更多缺陷，而且无法在一天或是一周之内解决它们。你可以把它们的优先级和严重程度设为‘低’，可我们的客户却不这么认为，他们会非常生气。然后你们就会给我施加更多压力，让我在不到4周的时间内发布一个小版本。问题在于，我们是不是必须要满足这个交付日期？”

蒂姆点了点头：“我们不能错过这个日期。”

丹回复道：“OK，我们要这么做。咱们不调整优先级和严重程度，而且可以在发布版本说明中加入一部分，命名为*已知问题*。我们可以排定这些问题的优先级，并让客户知道我们将在何时修复它们。你们觉得这样做怎么样？”

三个人讨论了丹的想法，他们同意采纳“已知问题”的做法，而不是改变待解决缺陷的优先级和严重程度。又避开了一次缺陷降级游戏！

15.4.3 规划回顾

项目经理一定要在项目结束时举行回顾。即使你一直在举行中期回顾（请参见8.2节），也要保证在项目结束时举行回顾。中期回顾不设推动者不会有太大问题，不过还是应该为最后的回顾寻找另外的推动者。项目经理和团队对于可交付物和项目工作过于了解了，以至于项目经理很难作为推动者推动回顾。

回顾既不是“经验教训”批评会，也不是对项目的盖棺论定。它是一个结构分明的会议，其目的是要回顾项目的进展过程、人们有哪些经验教训、他们在这个项目中工作时的感觉如何。经过用心设计和推进的回顾，可以为下个项目节省好几周的时间。

德比和拉尔森［DL06］将回顾分为5个步骤：

(1) 准备阶段
(2) 收集数据
(3) 深入讨论
(4) 行动决策
(5) 结束回顾

如果你曾管理过持续长达三个月甚至更长时间的项目，我强烈推荐你花上一整天的时间反思并分析刚完成的项目。如果上个项目团队的大部分人要一起参加下一个项目，就更应该这么做。没错，这要用去参加这个项目的每个人一天的时间。更长的项目甚至需要时间更长的回顾。

一天的时间看起来很长，尤其是项目团队超过20个人，而且有两个或两个以上地点的人参与的情况下。以团队为小组安排小规模的回顾也是可以的，不过要把所有的团队集中到一个地点。可是，如果团队拆得越零散，收集到的数据可用性就越低，从而项目或工程能从中得到的好处也就越少。

如果项目经理管理的项目规模很大，或是管理多地点项目，又该如何与项目团队中的每个人进行回顾呢？

应该小心行事。要是项目经理还希望从项目中收获一些东西，回顾还是要做的。首先，看看能不能把所有的人聚集到一个地方进行回顾，这个地方不属于任何团队所在的地点［Ker01］。在各个团队所在地点举行中期回顾时，项目经理会使用某些手法来推动回顾活动，这些手法此时同样适用。你还可以让管理层解决某些由于跨团队造成的问题。让每个团队推选一位代表，展示他们的经验和教训。还要让这些选举出来的代表分享他们的经验和体会，同时考虑由于跨地点造成的问题（这些不是管理层面的问题）。

还有一种变通方案：与所有的团队进行虚拟回顾。设置网络摄影机，让每个人都能看到各个团队的房间。使用wiki收集信息，因为wiki允许多人同时书写，同时每个人都能看到其中的内容。多准备几条可用的电话会议线路，这样就可以一组人共同发表意见了。还可以使用Cardmeeting.com让人们聚在一起，并把想法分成几组。[①]

记住，有些问题存在于不同站点的某些人之间。管理层无法解决这些问题。项目经理要把这些人带到同一个物理地点上，再解决这些问题。

15.4.4　规划庆祝

人们需要某种仪式。就像每个项目都要有个启动仪式，在项目结束时要记得举办庆祝仪式。

聚会或庆祝不一定要花多少钱，或是要把每个人的家人都聚到一起。我参加过一些很棒项目结束后的庆祝仪式，只是在产品大规模发布的那一天准备了一些啤酒、葡萄酒和冷餐。我的一个客户有来自多个国家的开发人员和测试人员，在庆祝时，他们带来了各自国家的独特食品。公司提供了（不含酒精的）饮料、饼干，还有冰激凌作为甜点。有个小项目团队决定用一个下午的时间去玩激光野战游戏，进行庆祝。

庆祝必须要让项目的参与者们感到舒服。如果项目经理希望让团队参与到更大规模的庆祝活动中，也是可以的，但是这就不是针对项目完成的庆祝活动了。

如果项目经理愿意在庆祝活动上说几句，不妨概括一下每个人在项目中的付出。我曾经说过类似下面的话（其中的姓名已经更改）。

“把大家聚在这里，享用冰激凌和饼干，是为了庆祝4.1版本的发布。我们干得很不错。我知道大家已经在回顾活动中表示出了对彼此的欣赏。在此，我想再多说几句。

“杰瑞德，你的幽默感总让我们可以在疲惫时打起精神。玛丽安，你总是能让我把会议大纲和记录做得正确无误。帕特里克，我能从你敲键盘的方式中感受到你的情绪，这让我很惊讶，而且也是一个很好的信号，让我知道何时干扰了你的正常工作。比尔，你总是能找到最令人惊奇的bug。下一次你来帮我审查我的项目文档，可以吗？塞勒斯，你能写得一笔好字。下次我们再用即时贴规划日程时，你能替我往即时贴上写字吗？

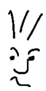
“咱们也别忘了没有正式加入项目团队的人们。虽然辛迪不是项目的正式人员，但她帮我保持了工作的条理性。谢谢你，辛迪，因为你在行政方面的热心支持，还能保证我总是准时拿到机票。

① 来自艾丝特・德比和戴安娜・拉尔森的一次私人谈话。

“就说这些。咱们开始享用甜点吧，让我们为共同完成的工作而自豪。我们真地干得非常棒！”

就是这样，赞扬每个人的优点。而不是仅仅说“干得不错，同志们。”那是毫无诚意的感谢。当你对大家说一些针对他们个人的感谢话语时，他们会觉得你在整个项目过程中都在留意他们的工作，而你也是这么做的。

如果项目经理打算多说几句，而不仅仅只是“让我们庆祝产品发布吧。我们这次干得不错”，那就把要说的话写下来，再进行练习。你甚至可以做一些笔记。为发言做准备的行为，会让团队觉得你从人的角度很关心他们。如果你以后再跟这些人合作，他们会记住这些，并用信任和尊敬来回报你。



小乔爱问……

失败的项目值得庆祝吗？

当然，至少要庆祝一下项目结束了。

在认定项目失败之前，要安排一次回顾。我发现：项目失败的原因常常来自管理层——包括出资人、高级管理层，甚至是项目经理。有时，出资人会在项目进行到一半时改变项目的总体目标。有时，组织需要项目采用阶段–关卡式的生命周期，却又希望项目可以像敏捷项目那样快速应对变化。有时，项目经理根本不收集任何测量数据，所以项目团队也根本不知道自己现在的工作状况。项目“失败”有很多原因，却很少是因为技术人员能力不足，无法完成技术工作。

15.5 取消项目

取消项目也是一种结束项目的方式，而且这会让参与项目的每个人都很不情愿。有一个开发人员对某个项目非常痴迷，即使在被公司解雇之后，他仍用自己的时间回来完成了那个项目！有人可能会觉得他是一个勇于奉献的热心人，也许是这样。但是他对项目的继续工作——即使是“免费”的，仍然占用了公司的时间。

如果你的公司决定取消项目，那就准备中止这个项目吧。[①]下面这些方式可以让项目工作停下来。

(1) 首先，向参与项目的人解释项目的取消原因以及对他们的影响。他们想知道接下来要做

① 请参见http://www.jrothman.com/Newsletter/kill-cancelled-projects.htm。

哪些工作。

(2) 感谢团队每个人为项目付出的努力。如果团队人员很少，可以在宣布取消项目的会议上向大家表示感谢［RD05］。对于人比较多或者是工程团队来说，让子项目经理或技术带头人去感谢他们的团队成员。

(3) 给人们时间，让他们先理清手上的事情，再开始新的工作。这可能包括签入之前签出的代码，并注明目前的代码状态，或是注明哪些设计正在讨论变通方案，也可能是要说明哪些测试已经执行、哪些没有执行。整理工作与收尾工作不同，我推荐用不到一天的时间来完成这个步骤。如果项目经理使用敏捷生命周期，这可能用几分钟、最多一小时就够了。对于其他的生命周期来说，把这个步骤限制在一天之内完成吧。

(4) 取消与该项目相关的所有定期会议。人们不再为这个项目的相关会议安排时间后，他们就可以为新的工作安排其他时间了。

(5) 找一个人专门处理取消项目带来的一些不可回避的问题，最好是某个管理层级比较高的人。如果某个搞技术的人知道了项目信息之后，他很有可能再次从事项目的某些工作。要是指派一个经理来处理这些问题，这位经理大概不会再去做这个项目了。

(6) 如果要取消的项目已经开始了一个星期甚至更多的时间，那就得花时间去做项目回顾，然后看看人们从项目中取得了哪些经验教训。

(7) 当人们整理完各自手上的工作之后，尽快让他们投入到新项目的工作中。

取消项目并不好玩。不过要是项目经理可以干净利落地取消一个项目，你就不用再折腾一次了，而且你也能帮助公司尽快投入到下一个应该做的项目之中。

铭记在心

- 将精力放在中期里程碑上，很容易引发“穿越沙漠综合症”，一定要避免这种情况。
- 在项目结束时，一定要做回顾，即使已经在项目中进行过中期回顾也要如此。
- 如果项目经理必须取消一个项目，那就取消吧。要取消得彻底，不能半途而废。

第 16 章

管理项目组合

16

这是一本关于项目管理的书，为什么要专拿出一章来讲项目组合管理呢？当我与其他项目经理一起工作时，他们会说“我的管理层搞不清楚他们想先完成哪个项目”，要不就是“我的管理层现在想要所有的东西”，还可能是“我的管理层很晚才作出有关项目的决策，所以我启动项目也晚了”，或者就是“我没办法让人们只做我的项目的工作，他们总是在同时处理很多任务。”

管理项目组合或是帮助管理层管理这些组合，可能是项目经理的一种求生技能。要是你的管理层可以制定企业（或是事业部）的战略，然后告诉你什么时候做哪些项目，不妨跳过这一章。如果管理层不是总能及时做出决定，这一章会有所帮助。

项目组合管理包括三个部分：构建项目列表［RD05］、评估每个项目、制定关于项目人员和资金的决策。有了组合之后，就可以根据每个产品正在执行的待办事项列表［Sch04］来管理项目组合了。待办事项列表可以帮项目经理更频繁地启动和结束更小规模的项目，从而提高产出。

16.1　构建所有项目的组合

哪些项目是活跃的、哪些项目应该活跃、哪些项目计划在何时启动，这些不是每个组织都能一清二楚的。第一步就是构建项目组合档案。

收集部门中所有的项目，形成列表。注明管理层认可的项目启动时间、认为项目应在何时启动，以及项目的预计结束时间。将这些项目按月放在网格中。[1]如果你选择用即时贴贴在墙上，看起来可能跟图16-1中的样子差不多。当然，你可以用电子表格或是网格线，让这个组合看起来更好看。可如果要想改变它的话，使用技术含量最低的工具，你可以很容易地进行调整。

有了项目组合之后，项目经理就可以从定量和定性两个角度来评估其中的项目了。

① 请参见http://www.jrothman.com/weblog/2006/03/courage-required.html。

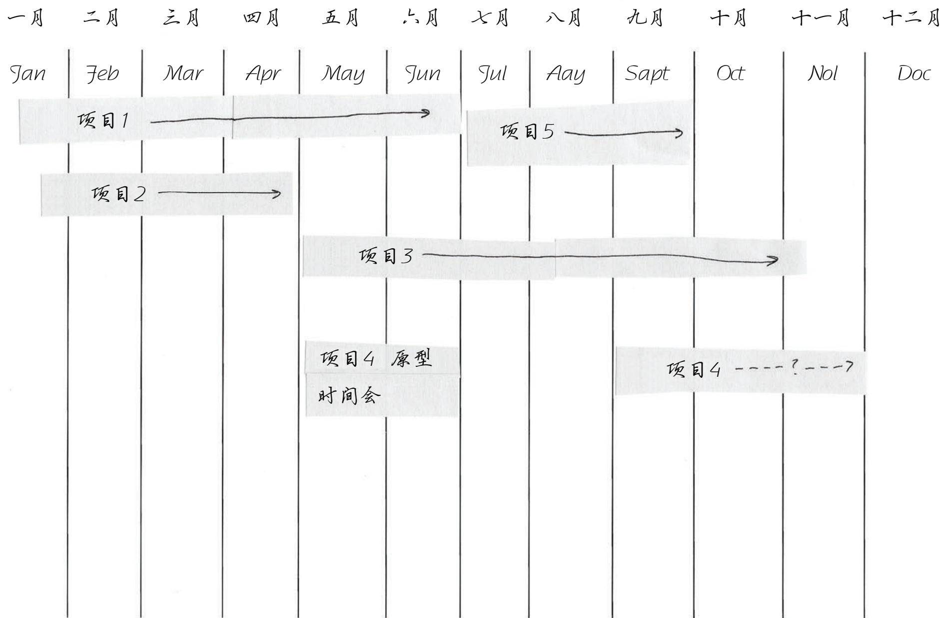


图16-1　项目组合档案

16.2　评估项目

要是对项目的起始和结束日期已经有所了解，即使是期望的起始和结束日期，你也可以开始评估这些项目了。你会希望从定性和定量两个角度进行评估。

16.2.1　定性评估项目

为了定性评估项目，可以问下面这些问题：[①]

- 这个项目与其他项目之间是什么关系？
- 实施这个项目的战略原因是什么？
- 完成这个项目有哪些战术上的收益？
- 为了使项目成功，我们是否提供了足够的资金？
- 为了使项目成功，我们是否提供了足够的人力？
- 我们是否知道这个项目成功的标准？

① 请参见http://www.jrothman.com/Papers/Cutter/Projectportfolio101.html。

如果不能为上面这些问题找到答案，也许该想想这个项目是否应该存在于项目组合之中了。有些时候，因为人力不够，公司会将某些项目保留在组合档案中，但是要弄清楚何时启动这样的项目，防止它们总是处于人力不足的状态。

16.2.2　定量评估项目

为了定量评估项目，可以问下面这些问题：

- 何时可以看到该项目的财务回报？
- 该项目的收益曲线是什么样子？
- 该项目的客户获取曲线是什么样子？
- 何时能见到该项目的当前客户保持率？
- 期望的客户增长曲线是什么样子？
- 何时能见到该项目降低运营成本？
- 期望的运营成本曲线是什么样子？

我的很多客户认为，有了这些问题，就足以针对项目组合做决策了。如果管理层还想要更多数据，可以参考［Coh06］，其中有一节专门提到了如何判断项目的财务价值。

项目评估完成后，你就可以对它们拿主意了。对项目组合中可能的项目做决定并不简单，这就是资深管理层能拿高薪的原因，对吧？还有，不对项目做决策也是一种决策，即使做出了有问题的决策，只要还能调整回来，相对不做决策来说，这样做的成本反而更低。

16.3　决定现在为哪个项目提供资金

使用定性和定量的问题，可以让项目经理跟人讨论现在为哪些项目提供资金，哪些项目可以稍微等等。有些项目只有等到公司钱多得没处花的时候才能给予资金帮助。不过管理层可能会在一些项目中晕头转向，弄不清楚应该先为哪些项目拨款、哪些稍候，以及何时应该改变这些决策。

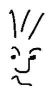
此时，使用敏捷生命周期（请参见附录A.4节）、构建产品待办事项列表（请参见16.6.1节）可以助你一臂之力。如果是迭代式开发，而且总是先开发优先级最高的需求，项目经理就可以在迭代结束后调整各个项目的优先级。（我不推荐这么做，不过你可以这么做。）

16.4　对组合中的项目进行排序

你已经评估过列表中的项目了，现在可以对它们进行排序。如果你知道先做哪个项目、其次

做哪个、再做哪个，就可以按照这个顺序安排人员。

对项目排序，类似于对项目的驱动因素排序（请参见1.4节）。某种程度上来看，一个项目的确要比其他项目更重要。如果管理层无法下判断，可以把所有的项目列在表格中，在各栏的栏头提出各种问题。跟管理层会面，一起讨论这些问题的答案。完成之后，你就有了一个项目的排序列表。你会知道何时适于开发哪些项目。有了排好顺序的项目组合之后，你就可以尽快开始启动某些项目了。



小乔爱问……

如果两个项目的优先级相同，我该如何决定？

这是一个经常发生的组织问题。你有两个项目，每个项目的侧重点不同，你也希望同时为它们提供资金，可是没有足够的人力安排到这两个项目中。二者的类似程度也不足以作为同一工程的不同部分。现在该怎么办？

应该回过头去看看组织的使命。如果两个项目不同，各自都需要同样的人力，你的资深管理层应该明确说明这些项目的使命。一旦他们确定了项目使命后，应该选择哪个项目也就很清楚了。

16.5 尽快启动项目

通常来说，造成项目启动时间过长的原因包括决策、生命周期选择、需求定义这三方面。作为项目经理，你可能无法解决所有的这些问题。不过如果你能识别出问题，并指派别人去发现原因、寻找解决方案，也许你能更快启动项目。

项目相关决策时间过长

要是项目看起来总是启动不了，你的管理层可能在决策方面遇到了问题。要确保你的决策过程包含如下步骤。

(1) 定义期望的产出，比如排定项目的顺序等，这样每个人就可以知道各个项目相对的重要性。

(2) 确立决策的边界，比如项目章程分析，还要指明谁会做出最后的决定。如果有人能够拍板决定项目何时启动、安排哪些人力，项目经理要与此人一起讨论。如果你跟一个委员会一起讨论，要是某个委员会成员对一项决策投反对票该怎么办？知道谁会做最后决策，是快速决策的关键。

(3) 提供多种方案，让人们可以从中选择，进行决策。我遇到过一些管理人员，他们不愿意做二选一的抉择（做这个项目或是那个项目），所以他们拒绝进行决策，不想认可这种二选一的方案。“3的法则”［Wei85］可以让人们认识到不一定只有非此即彼的选择，要达成决策，可以有很多种方式。而且还能让人们发现彼此想要的东西。有一个管理团队，几年之内都无法就任何项目的相对重要性达成一致，后来他们认识到不一定必须做出某种唯一的判断，只要决定何时应该完成哪个项目就可以了。看清这一点，他们也就不必让人同时为多个项目断断续续地开发原型了，而是将全职的人力投入到某一个项目上。

(4) 从多种选择方案中挑出一种，包括识别出你和管理层如何做出决策的条件。从多个项目中做选择并不容易。请参见16.2.1节和16.2.2节，提出问题，对项目进行评估。

(5) 执行决策。即使项目经理已经成功排定了项目的顺序、确定了决策的边界、识别出了多种选择方案并选定了一种，如果没有人说：“好，启动这个项目吧！”，它还是不会开始的。过上两到三个星期，当一个新项目突然从天而降要你接手，你就该感到惊讶了。要是你对面临的项目感到讶异，而且觉得项目都要启动了，自己却还一无所知，就要赶紧跟管理层讨论，让他们告诉你何时讨论项目相关的决策。项目经理不必参加所有的会议，但你得知道会议的结果是什么。

资深管理层可能难以得到预先期望的战略性讨论结果。你可以将此视为无法在多个项目中做决策，也许还会看到6.6节和6.7节中讨论的日程安排游戏。

发生这样的情况，可能是因为有些管理团队无法定义他们的决策边界。有些管理团队看不起有多少备选方案，更多管理团队可能很难从众多方案中作抉择，因此倾向于为所有的项目提供部分资金，而且会认为：让所有的项目都取得一些进度，这要胜过只资助部分项目。

如果你的管理层无法决定先启动哪个项目，项目经理可以帮忙。可以尝试两相对比的技巧（请参见8.3节）：对比两个项目，看看你更需要哪一个？两两对比所有的项目，最后就可以产生一个相对的项目排序列表了。

16.6 使用产品待办事项列表管理新功能需求

处理新功能需求是项目管理工作的一部分。如果你落入了任何日程安排游戏的虎口之中（特别是6.11节中的游戏），就该跟管理层好好讨论讨论如何定期处理新功能需求了。

16.6.1 构建产品待办事项列表

产品待办事项列表包括两个部分：这个发布版本要完成哪些功能，未来的发布版本要包含哪些功能。

不断变化的需求请求列表，也就是产品待办事项列表，与管理当前版本的需求不同。如果项目经理承诺客户，要在当前版本中满足某些需求，客户就会指望你到时交付这些功能。而且，不管项目经理是否要保证当前的迭代不变更，或是保护当前版本不变更，你的客户总是对未来的功能集合有潜在的要求。不断变化的列表可以隔离开当前版本与未来版本的需求。

产品待办事项列表是一个按发布版本排序的需求列表（或是未来可能成为需求的东西）［Sch04］。

列表中的需求不一定是完全成型的有效需求。它们可以是用户需求，或者是对于稍后讨论某个需求的承诺。但是其中的文字要能让开发人员明白背后的含义。图16-2就是一个按季度分类的类似列表。除了几个已命名的缺陷外，这个列表中没有多少条目看起来像是需求。注意第一季度那一列中的黑线。黑线上方的所有任务都要在第一季度完成，黑线下方的是在第一季度中可供协商的任务。如果项目团队不理解黑线下方的东西，或是无法在第一季度完成它们，这些“需求”可以挪到第二季完成。

图16-2　产品待办事项列表

保留产品待办事项列表（有人更喜欢称之为*产品路线图*）可以在多个层面上起到有益作用。扩展视图中可以包括4～6个季度的信息，因为你预计变化会频繁发生。适中的视图包含3到9个月的信息，因为你预计了现在这个季度之后，很多信息会变化。短期视图会包括0到3个月的信息。

根据实施生命周期的不同，项目经理不是预期变化就是管理变化。

使用敏捷生命周期时，项目经理不必关心功能需求在当前迭代结束后是如何经常变化的。作为项目经理，你要做到的，是保护当前迭代的工作内容不发生变化。当前迭代一结束，任何东西都可以变化。可能只有功能的规模和持续时间的变化会让出资人或管理层对于在哪个迭代开发哪些功能改变主意。为了快速估算待办事项列表中条目的工作量，可以使用规划扑克（请参见5.1.9节）。

如果使用增量式生命周期，特别是用时间盒限制需求收集和架构相关工作之后，只要开始开发成块的功能，项目经理就可以接受变化了，即使这些成块功能的持续时间不同也没有关系。然后，出资人和管理层就可以根据功能的大小、持续时间、特定人员能否参与来作决策了。

要是使用的是迭代式或增量式生命周期，项目经理可以使用季度待办事项列表，不过要保证发布版本的时间相对短；要么就修改生命周期模型，在原型或编码阶段开发成块的功能。

16.6.2　管理待办事项列表

基于组织不同的工作方式，项目经理可以与工程团队或是管理团队定期讨论，管理待办事项列表。（如果存在产品负责人或是产品经理，这个人会决定产品待办事项列表中有哪些内容，并推动关于待办事项列表的讨论。）在讨论中，项目经理应视需要调整待办事项列表中的功能排序。在敏捷生命周期中，这意味着在下个迭代开始前，项目经理要再次评审产品待办事项列表，并重新排序，从而为下个迭代敲定待办事项列表的内容。

如果使用顺序式生命周期，但是采取按功能逐个实现的方式，而且将发布的周期控制在3到6个月之内，项目经理可以每个月讨论一次产品待办事项列表。在规划下个版本之前，要多几次讨论，并在下个版本工作开始之前敲定最终的功能列表。

项目经理可以随时改变产品待办事项列表，调整各条目的优先级。你只能保护一个迭代涉及的待办事项列表内容。当迭代开始后，项目经理就不能改变这个迭代中待办事项列表的相关工作内容了。未来任何迭代或发布版本的内容可以随时变动。

16.7　组合管理答疑

你已经尽力了。你为每个产品都制定了项目列表和季度待办事项列表，而且还管着两个被管理层视为最高优先级的项目。你的人手有限，也不想让他们全部同时做两个项目的工作，可是别无选择。让人们同时着手多项任务会造成多少影响？现在该看看了。

16.7.1　管理从事多个项目多个任务的情况

在5.4节中，我曾告诉过你不要允许人们同时着手多个任务。如果速度对你真地很重要，那就不要允许出现多任务的情况。但是速度不是唯一的变量，有时多任务可以给组织带来好处。

让我解释清楚。允许（甚至去鼓励，这就更差了）人们同时处理的项目和任务数越多，项目的持续时间就会更长。但是，有时出于收益、客户体验或是保留某个客户等角度考虑，公司会让你改变项目的优先级。经过深思熟虑、仔细权衡，你可以应对多任务的情况。把人移走之后，完成这个项目的好处可能就得不到了，但是你会因为完成另外一个项目而受益。

假定你在管理一个工程，而且两个月之内无法向客户发布产品。整个工程的工作在取得进展，而且有可能在截止日期来临之时交付。假设有一个开发人员必须去为一个现有客户修复严重的问题。你也知道，没有了这个开发人员，工程就无法按时发布产品了。作为项目经理，你不想让此人去修复那个严重问题。但是作为一个公司内相对资深的管理人员，你认识到：让一名高兴的客户变得不高兴，这可不是什么好主意。你该怎么办？

只要你不会把项目团队的任何成员在项目之间呼来喝去（请参见6.6节），不造成“认知过载（cognitive overload）”的情况，[①]不妨先评价下各项工作任务的相对价值。你可能认为：保证已有客户的满意度优先级最高。（你的上下文可能与利润相关，而不是客户。）你也可能认为准时完成工程最重要。不过，不管你打算怎么做，要下定决心。让两个工程各占用开发人员的部分工作时间，这样一来，哪个工程都无法更快完成。

16.7.2　说服管理层“切换上下文”是个坏主意

管理人员，特别是高层管理人员，都不认为“切换上下文”会浪费时间，因为他们每天都是这么做的。高层管理人员经常要同时处理多个项目，大部分人都在等待别人提供信息。但是技术人员的工作方式完全不同。在大部分时间内，技术人员都不会处在“等待”状态，却可以一直高效地从事项目相关的工作。高层管理人员无法理解，也许是遗忘了技术人员要进行的工作与他们的工作实质上完全不同。

为了说服管理层“切换上下文”是个坏主意，项目经理要使用他们的表达方式。首先，说明同时完成多个技术任务的成本，为后面打下基础。接下来，项目经理要确保自己有项目组合，这样你和你的管理层就可以讨论各个项目的相对优先级了。当你理解了多任务的成本和价值之后，设计一下工作方式，并定期告知管理层各项工作的进度。

① 请参见http://seattletiems.nwsource.com/pacificnw/2004/1128/cover.html。

说明同时完成多个技术任务的成本

管理层会准备同时在多个项目上工作，而且处在等待几个项目进展的状态。而且，管理层一般都有助理，帮助他们管理项目，从事项目相关工作，安排这些管理人员的时间。可技术人员没有助理。对于多任务消耗的成本，他们的体会可比管理人员强烈得多。

在图16-3中可以看到两个任务的图，完成每个任务都要用一周时间。如果你一次只做一个任务，直到把它做完为止，任务1会在第一周结束时完成，任务2 在第二周结束时完成。

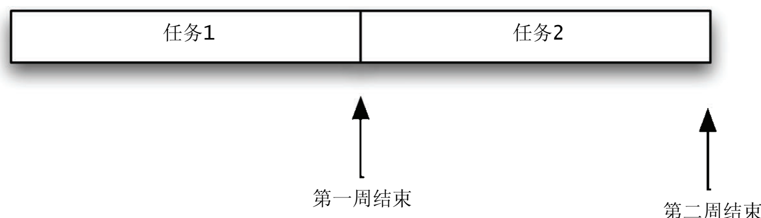


图16-3　两个任务，不同时处理

但是请看图16-4，这就是同时处理多任务发生的情况。假设爱丽丝被指派完成这两个任务，先在任务1上花去一整天，第二天处理任务2。而且，我们假定爱丽丝不会受到其他干扰。

任何一项任务，爱丽丝最快也要将近第二周结束时才能完成，届时她会完成任务1。在第三周开始的时候，爱丽丝可以完成任务2.

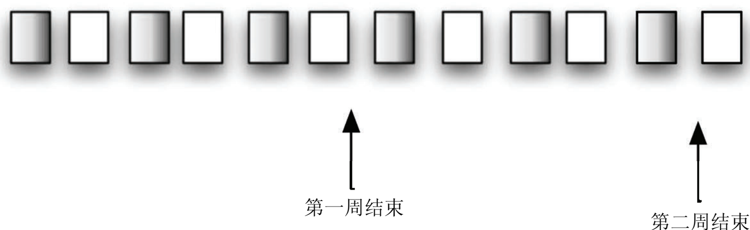


图16-4　两个任务，同时处理

爱丽丝手上的任务（问题、其他项目，不管是什么）越多，她要完成任何任务花费的时间就越长。这是因为爱丽丝每次在切换任务时，都会发生成本。对于开发人员来说，成本体现在如下方面。

- 中止你手上正在做的工作。中止的成本主要是时间，包括你注明当前工作进度、保存目前工作以及诸如此类事情的时间。你并没有停止思考正在做的工作，可当你停下来接电话或是回答问题的时候，就会产生中止成本。如果你已经进入了“流”的状态，这个成本会出奇地高。

16

- 向外切换正在进行的工作任务。向外切换就是把之前一直在做的工作从脑海中清除掉，为将要向内切换的工作空出地方。如果你处于“流”的状态，或是正在集中深入思考，这大概要用去5到30分钟的时间，有时会用去更久。
- 向内切换新的任务。向内切换要基于工作的复杂度，以及上次接触该任务到现在的间隔时间。任务越复杂，间隔时间越长，要一起讨论的人越多，向内切换就要用去更长时间。
- 等待别人停下手里的工作，与你讨论你的新任务。在你必须要接受一个新任务时，要等待别人一起讨论，这会形成倍增效应。
- 将原来的任务再切换回来。考虑到任务的复杂性，要再把脑子转回来，可能要用数秒到一小时不等。（从项目管理的角度来看，缺陷在这时悄然而入，因为很难记住之前中止时的所有细节。）

明确说明多任务的成本之后，项目经理要确保已经了解了各个项目的相对优先级。跟管理层复审项目组合。如果你已经问过定性和定量的问题，是否仍然可以得到同样的答案？如果是的，那就一切照旧，你的管理层会同意不为新任务安排人力。不过也许情况已经不一样了，现在改变优先级也许是正确的做法。要说明你需要多久才能将人力从一个项目调配到另一个项目，并让管理层表示同意，还要说明何时再次回顾这个决策。然后，项目经理要保证在项目中使用增量式或敏捷生命周期，这样你才能有最大的灵活性。

16.7.3　如何对多任务说“不”

你和你的管理层都认识到：原本想要做的工作并不比现在的工作更有价值——至少现在是这样。可不管是你还是管理层，都很难对更多的工作说“不”。

有时，说“不”并不合适。有时，项目经理觉得自己像个恶人，因为你想说“不”。在有些组织中，这不仅仅是不合适的问题，更像是一种职业自杀行为。不过，你有很多种对更多工作说“不”的方法，同时不必用“不”这个字眼。我用过下面这些方法，而且很有效。看看哪个对你有用吧。否则，你就会经常遇到6.12节中那样的日程安排游戏——“我们不能说‘不’。”

“现在不行”，再给个新日期。你可以说：“我现在不能做那个，不过团队可以从4月份开始。我们可以从这个日期开始，到那个日期结束。”这可以让管理层看到你其他工作的优先级，他们也能知道，如果硬要你开始新的工作，会让你中止哪些工作。

“这些是我现在做的工作——我应该停下来哪一项？”这里要注意，说话时态度一定要好。如果你使用很讽刺的语气，反而不能帮到自己。

当项目经理向其上级（或是上级的上级）展示所有的工作列表时，要说明优先级：“我们在为劳拉完成这个项目。汤姆和贝蒂在为西德妮完成那个项目。而且，我们还要给艾伦做那个项目。

我的团队不能再接更多工作了，也没有更多人手。你想让我们暂停哪个项目呢？”

与上司一起制定产品待办事项列表。有时，管理层让你完成更多工作，是因为他们不知道你们有哪些工作成果，或是不信赖你们团队完成的工作。此时，你可以使用有时间盒限制的迭代，并与上司一起建立产品待办事项列表，让管理层看到你们团队能够在什么时候有哪些产出。请参见16.6.1节。

使用项目组合排定工作优先级。如果更高层的管理人员让你多做一项工作，并导致多任务的情况。可以尝试这样的谈话：“吉姆，我知道你有五六个正在进行的项目。可要是我们同时处理多任务的话，要完成工作就得用去更多时间。而且我从产品经理处得到的反馈是，这个发布版本非常重要。咱们一起看看项目组合，排定优先级吧，这样我也可以理解你的优先级是怎么样的。”

带上你的产品待办事项列表，把工作拆成多个迭代或是相对较小的发布版本（不超过三个月)。你可以规划一个项目组合，说明你要在什么时候为哪个项目完成哪些工作。我曾用过这样一个技巧：用一两周的迭代完成每个项目收尾时的工作。你不必非得同时进行多个项目的工作，使用一周的迭代完成某个项目的工作，这要好过多任务的情况。

多任务请求是有风险和后果的。可以试着指出多任务请求在业务上有哪些风险。“约翰，我们可以做那个工作。如果我们真那么做的话，技术文案的工作就会落在这个版本发布之后了，因为开发人员离开去做其他项目。我不知道你还记不记得，去年，那个非常重要的客户给我们的大人物打电话，说我们没有记录文档的那个工作流耗费了他们不计其数的金钱和时间。我担心类似情况再次发生。咱们能担得起这个风险吗？”

“你什么时候需要这些功能？”有时，管理层是想让你往项目组合中加入更多工作，而不是马上开始。不妨问一问：“你什么时候需要这些功能？我可以把它们加入到下个迭代要完成的待办事项列表中。”如果是一个全新项目，询问组织何时需要这个项目，可以这么说：“我可以先把这个项目放在这里。等罗恩完成手上的项目就马上开始，可以吗？”

如果你已尝试过上述方法，你的出资人还是不愿意接受“不”，你就等着项目失败吧。这样的话，想想是不是到了该离开的时候了。请参见7.7节。

铭记在心

- 使用产品待办事项列表，不管你采用什么样的生命周期模型。
- 制定项目组合，以从视觉上了解所有项目的情况，不管这些项目是在进行中，还是刚做计划。
- 学着如何说“不”。

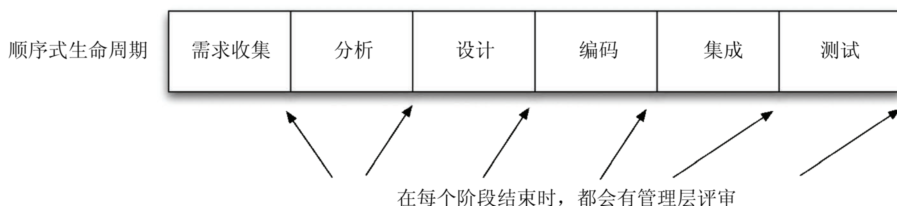
附录 A

关于项目生命周期的更多详细信息

A

A.1 顺序式生命周期：瀑布式或阶段–关卡式

如果你的项目没有什么技术上和日程上的风险，团队人员也很稳定，需求变化的风险也很低，那就可以考虑使用顺序式生命周期。也就是说，如果你的项目时间很短，人数很少，需求也很清晰（比如修复前一版本的问题），就可以选用顺序式生命周期（请参见图A-1）。请参见第11章，了解如何在顺序式生命周期项目中度量数据，从而深入了解项目的真实状态。



图A-1 类似于甘特图的顺序式生命周期示意图

A.1.1 顺序式生命周期明确解决的风险

顺序式生命周期可以解决下面这些明显的风险：

- ❑ 功能集合
- ❑ 知道何时完成哪些工作
- ❑ 成本风险

顺序式生命周期为功能集合相关的风险做了“优化”。因为在采用顺序式生命周期的项目中，需求一开始就定义好了。

在顺序式生命周期中，从日程中可以很清楚地了解到现在应该处于哪个阶段，所以在项目开始阶段创建一个日程很容易。（要创建一个有用的日程可就不仅仅是“困难”了，那几乎不可能。）即使是在阶段–关卡式生命周期中，就算签字允许一个阶段在前一阶段尚未结束时开始，也很容易知道项目目前理应处在的进展状况。而且，由于团队在项目开始时花时间生成了需求，所以要在项目一开始时，就可以确定在什么时候开发哪些功能。

使用顺序式生命周期，要确保使用基于可交付物的规划（请参见4.3.6节），将可交付物作为里程碑，而不是阶段的结束。如果将阶段的结束作为主要的里程碑，而没有逐步汇聚可交付物，项目经理也许要到测试阶段才能发现整个项目已经延迟了。

一直以来，人们对于顺序式生命周期管理成本的能力给予了过高的评价，因为可以将管理层的意见和重新估算的结果集成到项目计划中。在现实中，使用其他任何生命周期都更易于管理成本，因为更容易度量和查看项目的进度。

如果查看最早的论文［Roy70］可以发现，瀑布式生命周期原本打算用作带有反馈循环的迭代式开发方法。罗伊斯建议团队先反复分析项目，然后再实现，而不是试图从一开始就收集所有的需求，完成所有的分析。

谁还在用瀑布式生命周期？

看到敏捷相关实践和生命周期的蓬勃发展，我的同事们一直在说：“再也没有人用瀑布式生命周期了。”但是仍有很多项目还在使用。

我知道有些成功的项目使用带有反馈的顺序式生命周期。这些项目的项目经理将主要的里程碑作为收集反馈的机会。他们与管理层复审项目，从而获得了管理团队的反馈；不仅如此，有些项目经理还会主持中期回顾（请参见8.2节）。虽然这些项目耗时超出预计时间，但是确实发布了客户愿意使用的软件。有些成功的项目经理就使用了瀑布式生命周期。

但是，有太多使用顺序式生命周期的项目经理并没有利用各个主要里程碑或项目阶段来重新规划项目。如果你必须使用顺序式生命周期，要小心管理风险，因为恐怕很难看到产品随项目的进度发展。

A.1.2　顺序式生命周期掩盖的风险

虽然有人认为顺序式生命周期可以暴露下列风险，但实际上这些风险被这种生命周期掩盖起来了，直到不得已的时候才显露出来。

- 架构风险。很多成功的技术人士都认为，他们应该在项目一开始完整定义整个架构。而且他们相信，提供这样的架构能够降低架构方面的风险。

  一开始就定义完整架构（敏捷社区称之为“大规模前期设计”）的问题在于：无法预知架构是否有效，除非开发人员实现了某些功能。而且，由于架构是通过组件定义的，人们也会倾向于开发组件，除非等到集成阶段，他们无法获知架构是否能够正常工作。

- 测试风险。有些组织相信：由于顺序式生命周期在开发结束后有专门的测试阶段，所以测试只能在编码结束或进行到一半时开始。这样也不会出什么问题。

  正式的最终系统测试是在编码结束之后。但是，如果你必须使用顺序式生命周期，还是要在项目开始时就把测试和测试人员集成进去。请参见第13章了解更多信息。测试不仅仅是要测试代码，还包括通过需求评审和检查验证需求的过程，也包括通过架构评审验证架构合理性的过程，还有其他工作。任何工作成果都可以“测试”，这并非仅限于代码测试。

  如果要为受管制的行业开发产品，你需要进行正式的复核步骤。类似的项目要在整个过程中跟踪需求，还要交付测试日志和其他产出，以说明需求的符合程度。不要把检验（verification；产品测试，确保系统可以正常工作）和复核（validation；流程测试，以确保产品按照正确的方式开发）混为一谈。为了检验，项目经理要让测试人员跟开发人员紧密协作；为了复核，项目经理也许需要一个审核职能流程。请参见13.10节，了解QA团队能够提供哪些帮助。

  在顺序式生命周期中，开发人员要等到编码阶段才能提供反馈，这就是为什么顺序式生命周期会有那么高的日程风险。测试进行得晚，就会用去更长时间。项目经理要把这些多出来的时间加入到日程中。顺序式生命周期并没有提供太多反馈，如果你要求开发人员跟测试人员测试所有的工作成果和想法，那么开发人员就能够得到更多反馈。

- 日程风险。如果人稍微多一点、时间稍微长一点，使用顺序式生命周期的项目就很难按日程准时交付。主要原因在于提供给开发人员的反馈不够。因为一个想法在实现之前是很难验证的，开发人员不知道他们使用的架构是否合适，除非进入到项目的编码阶段。除了缺少给开发人员反馈之外，测试人员通常也只能测试最后的产品，所以他们很晚才能介入项目。如果开发人员遇到麻烦，为了按日期准时交付，出资人经常会占用项目测试阶段的时间。缺少测试人员和测试过程会减少给开发人员的反馈，从而降低开发人员的积极性，并导致项目延期，或是产生更多缺陷，有时二者兼有。

A.1.3 何时重新规划顺序式生命周期

项目经理可以利用每个阶段的里程碑重新规划项目剩下的工作。即使你从未用过波浪式规划，也可以考虑在阶段–关卡时或瀑布式生命周期中使用（请参见5.6节）。使用波浪式规划，项

目经理有机会每周规划一次，或是按其他你希望使用的时间长度。即使你只能在各阶段结束时重新规划，这也有助于把项目转向更为合理的方向。

顺序式生命周期会引诱你做预测

顺序式生命周期有着邪恶的诱惑力。在创建甘特图时，项目经理似乎很清楚什么时候会发生哪些事情，这就是在做预测。你向一个水晶球中看去，希望自己能够看见未来。

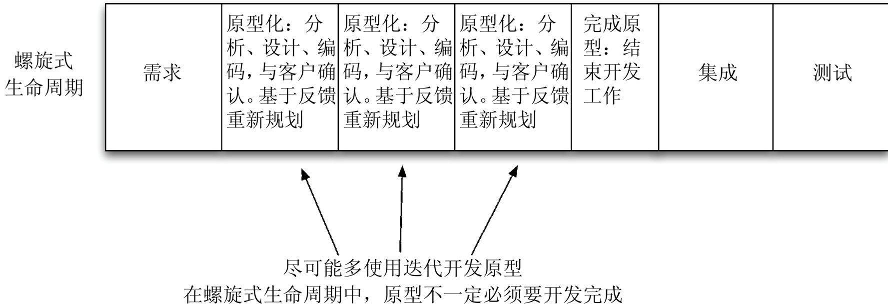
但是项目中布满风险。人们开始着手项目工作后，当初你小心翼翼设计的日程很容易就四分五裂了。因为风险的存在，项目的日程很难一早确定。而且，项目经理也无法预先获知这些日程。

当气象学家在预测天气时，他们会收集实际的天气数据，以改进预测效果。但是使用顺序式生命周期的项目经理无法收集项目或产品数据，除非进入到项目的编码阶段。到那时，再发现项目从第二周开始就已经偏离日程已经太晚了。

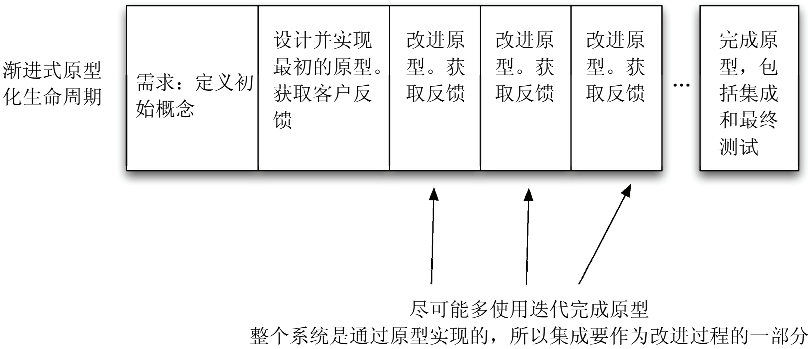
如果项目经理必须使用顺序式生命周期，要小心你和团队可能进行的预测行为，这些基于估算和文档的预测可能根本没有现实依据。而且只有到了项目后期才可能发现，到时恐怕为时已晚。

A.2 迭代式生命周期：螺旋式、渐进式原型、统一过程

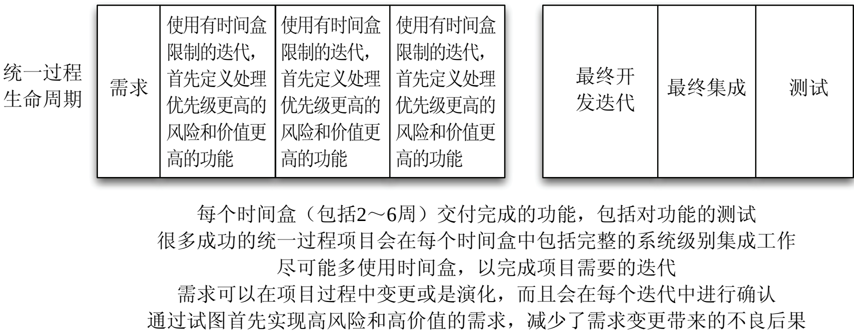
如果客户愿意跟你一起工作，而且你也能管理团队的原型化工作，要是你想看到多个功能在最后的产品上展现出什么效果，使用迭代式的生命周期会很有帮助。如果项目经理想先用原型化方式实现系统的一大块功能，再完成整个产品的开发，不妨使用螺旋式生命周期（请参见图A-2）。要是项目经理认为螺旋式生命周期无法应对集成带来的技术风险，可以使用渐进式原型生命周期（请参见图A-3）。图A-4中展示的统一过程［其中RUP（Rational统一过程）最为著名］对管理项目很有帮助，因为其中使用时间盒限制的迭代和集成能够帮助管理日程风险。



图A-2 类似于甘特图的螺旋式生命周期示意图



图A-3 类似于甘特图的渐进式原型化生命周期示意图



图A-4 类似于甘特图的统一过程生命周期示意图

A.2.1 迭代式生命周期解决的风险

迭代式生命周期会解决下列风险。

频繁变化的需求。因为每次只针对系统的某一部分开发原型，很容易加入额外的或是变更的需求。

技术（架构或设计）风险。通过早期的原型化工作，团队可以确定可供项目使用的架构或设计。

A.2.2 迭代式生命周期暴露的风险

迭代式生命周期会暴露如下风险。

- 日程风险。在真正的螺旋式或渐进式原型化生命周期中，结束项目可能很困难，尤其是

在团队长期以来一直喜欢从事原型化工作的情形下。对于统一过程来说，如果团队没有坚持使用时间盒或是无法在时间盒内完成每个迭代的工作时，项目很难结束。

- 成本风险。这些生命周期假定团队会先实现风险最高的功能，而不一定是产品最有价值的功能。可风险最高的功能并不一定是最有价值的。请参见9.3.1节。

A.2.3　迭代式生命周期与敏捷生命周期的不同

迭代式生命周期与敏捷生命周期的不同体现在如下方面。

- 在敏捷项目中，从始至终使用同样持续时间（一至四周）的时间盒迭代。严格的迭代式生命周期对时间盒长度没有标准限制，可以是两周、五周、三周，只要项目允许即可。（有些团队在实施RUP时，会选择标准大小的时间盒。）
- 敏捷生命周期努力在迭代结束时交付完整的功能，而且会基于上个迭代完成的工作改变下个迭代原本要做的工作。迭代式生命周期在迭代中不一定要完成某个功能，可能一个迭代的目标是要开发原型，而不是完成功能、测试系统或是评审代码。

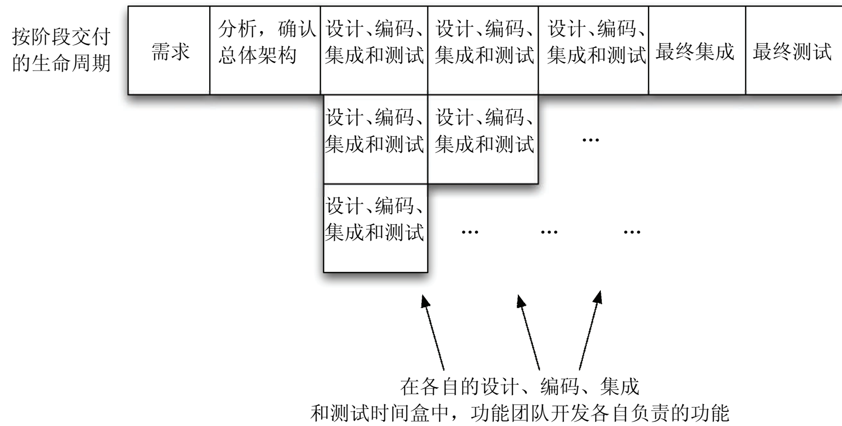
A.2.4　何时使用迭代式生命周期重新规划

对于下一个迭代来说，在当前迭代结束时重新规划是最有效的工作方式。如果你发现需要在迭代中期进行规划，你的迭代可能时间太长了，要么就是团队想往迭代中加入过多的工作。

在重新规划时，项目经理要想清楚：什么时候让开发人员完成原型化工作，并加入到发布版本代码中。可以考虑将迭代式生命周期与增量式生命周期，或是敏捷生命周期结合在一起，再做重新规划，这样一来，项目经理既可以享受到项目早期时原型化工作带来的好处，也能受益于项目后面完成的功能。

A.3　增量式生命周期：按阶段交付、设计决定日程

如果项目经理无法与客户持续取得联系，而且你可以创建功能团队，按功能逐个实现，使用增量式生命周期（请参见图A-5）非常合适。如果你的团队是跨职能团队，并且由他们开发完成功能，那使用增量式生命周期就能取得更好的效果。即使整个功能团队坐在一起，如果他们无法完成某个功能所有的工作（包括测试和文档编写），那就不能取得最好的效果。要组织团队使用增量式生命周期，应该考虑尽早获取稀缺资源（包括人力、机器等），然后早点儿在项目中实现需要稀缺资源的功能。如果稀缺资源已经分配到项目中了，功能团队还是完不成自己的工作，那他们可能就无法完成项目了。



图A-5　类似于甘特图的增量式生命周期示意图

A.3.1　增量式生命周期解决的风险

增量式生命周期会解决下列风险。

- 日程风险。团队在构建、测试和集成功能时，项目经理可以收集团队的真实进度，从而减少日程相关风险。
- 项目团队人员变更。如果有人离开了项目，此人的离开也只会影响一个跨职能团队，而不是整个项目团队。
- 需求变更。只要这些变更不会影响底层的产品架构。

A.3.2　增量式生命周期暴露的风险

增量式生命周期会暴露如下风险。

- 架构风险。在架构层面上，如果团队没有搞清楚应该先实现哪个功能，也许到后面某个功能会导致架构的变化。
- 需求变更。因为团队会随着工作进展完成功能，如果有人想改变某个已经开发完成的功能，团队可能要做更多工作。

A.3.3　何时使用增量式生命周期重新规划

可以考虑在每个功能实现完成后，进行重新规划。增量式生命周期很适合使用波浪式规划（请参见5.6节），这样一来，整个项目也就不用做太多重新规划了。

A.4 敏捷生命周期

敏捷生命周期最善于（如图A-6）应对日程、技术、需求和成本等方面的风险。它们总是使用有短时间盒限制的迭代来完成功能，而且经常是最有价值的功能。这些生命周期需要经常跟客户或是客户代理人打交道。（在为期一周的迭代中，需要客户每天都可以回答问题。）只要团队成员不必多任务处理多个项目的工作，不在时间盒中离开，而且可以致力于完成功能，使用这种生命周期，团队就很可能取得成功。

图A-6　类似于甘特图的敏捷式生命周期示意图

当我跟团队一起考虑转向敏捷生命周期时，他们经常面临这个问题："我们得先知道如何拆分任务，并做出适当的估算。现在我们不是这么做的，所以很难转向敏捷，是吗？"当然不是，只要项目经理开始为速度图收集数据，而且团队也知道他们要学习如何拆分任务，转向敏捷就不难实现。TDD（请参见13.3节）可以帮助团队学习如何从更小的角度思考。

A.4.1 敏捷生命周期解决的风险

敏捷生命周期会解决下列风险。

- 日程风险。因为项目经理只规划一个迭代中要完成的工作，要重新规划日程就很容易，也可以方便地知道需要哪些东西才能做到按日程推进项目。
- 项目团队变更。只要项目团队不在迭代之内变化，团队就可以自如应对人员变化。
- 需求变更。当然，如果某个需求的变化会修改架构，这是个问题，但是任何生命周期都面临这个问题。由于敏捷生命周期使用可灵活变动的待办事项列表管理需求变更，因而要想搞清楚这些变化的成本和价值就很容易。

- 成本风险。由于产品负责人可以在下个迭代开始前重新排定需求顺序（请参见16.6.2节），而且迭代时间相对不长，在管理层或是产品负责人变更规划前，项目就可以结束某个迭代工作了。

在敏捷生命周期中，项目经理的职责是保护迭代的工作内容，以及从事项目工作的人员状态。项目的需求可以在迭代之外不断演变。你也可以加入或去掉功能，但是任何迭代的工作内容是不能变化的。

除了迭代内容之外，项目经理还要保证迭代的人员不受干扰。在一个迭代中，所有团队成员必须全心投入到项目工作中，而且只能是这个项目的工作。那干扰怎么办？干扰只能在迭代结束后发生。没错，你可以随时中止一个迭代，处理干扰，再为下个迭代重新规划。要知道如何应对干扰，请参见8.11节。（如果团队成员是稀缺资源，你无法在迭代中保护他们，请参见5.5节。）

A.4.2　敏捷生命周期暴露的风险

如果项目经理需要管理日程、成本、需求和技术这几个层面的风险，敏捷生命周期是最佳选择。敏捷生命周期可以揭示出这样的风险：你的管理层不知道各个项目如何排定优先级。而且还会揭示出这样的风险：应该负责确认需求的人无法确认。而且，如果你的管理层不愿意保护项目团队成员和迭代的工作内容，项目经理就无法维持生命周期的正常运转了。任何生命周期都是如此。

敏捷生命周期对人和工作环境都有很多要求，如果工作环境不支持这种生命周期，这会变得非常明显。要想长期保持项目团队的组成，这就是一个风险。（如果你或者管理层不能言行一致，人们就会把你当成笨蛋［RD05］，最终离开团队。）

A.4.3　何时考虑使用敏捷生命周期

项目经理面对的风险越多，敏捷生命周期和敏捷实践就越能起到帮助作用。Scrum［Sch04］就是一种项目管理框架。你要时刻准备好保护迭代的工作内容，保证从事项目工作的人们不会被拉去做其他项目。由于敏捷生命周期使用短迭代，项目经理可以尝试规划三个迭代的工作，然后看看实际状况如何，再为项目剩余工作做计划。（我推荐三个迭代，因为项目经理和团队可能要用三个迭代时间来找到你们固有的节奏。）

A.4.4　何时使用敏捷生命周期重新规划

在迭代开始时重新规划，持续不断地重新组织产品待办事项列表（而不是这个迭代的工作内容）。

A.4.5　敏捷生命周期是基于反馈的

敏捷生命周期用反馈为团队提供项目的真实情况。项目经理不需要水晶球来看发生了什么，你可以看到、触摸、度量和感知实际发生了什么。

敏捷生命周期是唯一从头到尾基于反馈的生命周期。如果你有一个风险很高的项目，考虑使用敏捷生命周期吧，这样你从第一天就可以获得关于项目的反馈。

附录 B

术语汇总

关键路径（critical path）：由相关任务构成的最长的顺序任务链，对于完成项目来说，其花费的时间最短。

穿越沙漠综合症（Crossing the Desert Syndrome）：这是你认为永远无法完成任务时会有的一种感觉。团队集中全部注意力和时间，试图达成某个项目中间阶段的里程碑，此时会引发这种状况。他们即使完成了这个里程碑，项目仍然有很多工作要做。也许你一开始认为自己已经到达了绿洲，可是很快就会意识到，其实离得还远着呢。①

小石子（inch-pepples）：这是一些只需一两天即可完成的任务，它们的状态要么是完成，要么是未完成。

生命周期（life cycle）：项目经理带领团队完成项目工作的方式。

停车位（parking lot）：有些问题现在不想解决，但是又不能置之不理，它们就会被放到停车位中。

产品负责人（product owner）：他负责决定产品需要哪些功能且何时加入这些功能，同时负责管理产品待办事项。

工程管理（program management）：其意味着要协调多个子项目或是一系列项目，以达成某些特定的业务目标。

项目（project）：是指一个独特的任务或是系统化的流程，其目的是创建新的产品或服务，产品和服务的交付标志着项目的结束。项目都有风险，并且受制于有限的资源。②

项目经理（project manager）：负责向团队清晰说明“完成”的含义，并带领团队完成项

① 我是从橡树联合公司的杰克·奈维森那里第一次听到这个说法的。

② © 2007 R. 麦克斯·怀德曼，http://www.maxwideman.com；经许可后复制。

目的人。“完成”是指产品符合公司对这个产品的要求，也能满足客户使用这个产品的需要。

产品（product）：项目产生的一系列可交付物。

波浪式规划（rolling-wave planning）：为固定的时间安排详细日程规划的过程。进行中的规划会维护时间盒。

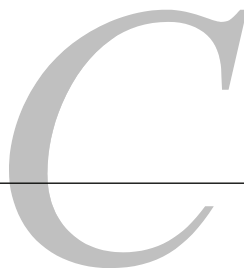
试探性开发（spike）：是一项纳入时间盒的任务，其目的是获取其他工作的信息。在极限编程中，时间盒中的工作会被抛弃掉［Coh06］。

技术债务（technical dept）：团队在之前的发布中没有完成的工作，拖延到当前产品，就会形成技术债务。团队任何没有完成的工作都有可能形成技术债务，比如设计债务、代码重新设计或重构债务、测试债务、开发债务，等等。

工作分解结构（Work Breakdown Structure，WBS）：是任务的一种组织形式，展示了它们之间的依赖、持续时间以及所有者。WBS的级别越高（在项目中出现得越早），项目经理对其了解得越少。WBS会随着项目进展而演化。

附录 C

参考书目



[Aus96] Robert D. Austin. *Measuring and Managing Performance in Organizations*. Dorset House Publishing, New York, 1996.

[BB96] Tony Buzan and Barry Buzan. *The Mind Map Book: How to Use Radiant Thinking to Maximize Your Brain's Untapped Potential*. Plume, New York, NY, 1996.

[BF01] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley, Reading, MA, 2001.

[Bro95] Frederick P. Brooks, Jr. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA, anniversary edition, 1995.

[BWe01] James Bullock, Gerald M. Weinberg, and Marie Benesh eds. *Roundtable on Project Management: A SHAPE Forum Dialogue*. Dorset House Publishing, New York, 2001.

[CB91] Allen R. Cohen and David L. Bradford. *Influence without Authority*. John Wiley & Sons, New York, 1991.

[CK02] Bret Pettichord Cem Kaner, James Bach. *Lessons Learned in Software Testing: A Context-Driven Approach*. John Wiley & Sons, New York, NY, 2002.

[Coc01] Alistair Cockburn. *Agile Software Development*. Addison Wesley Longman, Reading, MA, 2001.

[Coc04] Alistair Cockburn. *Crystal Clear: A Human-Powered Methodology for Small Teams*. Addison Wesley Longman, Reading, MA, 2004.

[Coh06] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall, Englewood Cliffs, NJ, 2006.

[Cov91] Stephen R. Covey. *Principle-Centered Leadership*. Summit Books, New York, 1991.

[CS98] Michael A. Cusamano and Richard W. Selby. *Microsoft Secrets*. Touchstone, New York, 1998.

[DeM86] Tom DeMarco. *Controlling Software Projects: Management, Measurement, Estimation*. Prentice Hall, Englewood Cliffs, NJ, 1986.

[DeM97] Tom DeMarco. *The Deadline*. Dorset House, New York, 1997.

[DeM01] Tom DeMarco. *Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency*. Broadway Books, New York, 2001.

[DL99] Tom Demarco and Timothy Lister. *Peopleware: Productive Projects and Teams*. Dorset House, New York, NY, second edition, 1999.

[DL03] Tom Demarco and Timothy Lister. *Waltzing with Bears: Managing Risk on Software Projects*. Dorset House, New York, NY, 2003.

[DL06] Esther Derby and Diana Larsen. *Agile Retrospectives: Making Good Teams Great*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2006.

[Gol97] Eliyahu Goldratt. *Critical Chain*. North River Press, Great Barrington, MA, 1997.

[Gol04] Eliyahu Goldratt. *The Goal, 3rd ed*. North River Press, Great Barrington, MA, 2004.

[Gra92] Robert B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, Englewood Cliffs, NJ, 1992.

[GW89] Donald C. Gause and Gerald M. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House, New York, 1989.

[Hal07] Payson Hall. Exploring project priorities. *stickyminds.com*, 2007. http://www.stickyminds.com/s.asp?F=S11953_COL_2.

[HT03] Andy Hunt and Dave Thomas. The art of enbugging. *IEEE Software*, 10(1):10–11, 2003.

[JAH02] Ron Jeffries, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed*. Addison-Wesley, Reading, MA, 2002.

[Ker01] Norman L. Kerth. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House, New York, 2001.

[Koh93] Alfie Kohn. *Punished by Rewards: The Trouble with Gold Stars, Incentive PLans, A's, Praise, and Other Bribes*. Houghton Mifflin Company, Boston, 1993.

[KS99] Jon R. Katzenbach and Douglas K. Smith. *The Wisdom of Team: Creating the High-Performance Organization*. HarperCollins Publishers, New York, 1999.

[McC96] Steve McConnell. *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, Redmond, WA, 1996.

[McC06] Steve McConnell. *Software Estimation: Demystifying the Black Art*. Microsoft Press, Redmond, WA, 2006.

[Mey93] Christopher Meyer. *Fast Cycle Time: How to Align Purpose, Strategy, and Structure for Speed*. The Free Press, New York, 1993.

[Moo91] Geoffrey A. Moore. *Crossing the Chasm*. Harper Business, New York, NY, 1991.

[MP06] Mary and Tom Poppendieck. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley, Reading, MA, 2006.

[Phi04] Dwayne Phillips. *The Software Project Manager's Handbook: Principles that work at work, 2nd ed*. IEEE/Wiley, Hoboken, NJ, 2004.

[RBS00] ed. R. Brian Stanfield. *The Art of Focused Conversation, 100 Ways to access Group Wisdom in the Workplace*. New Society Publishers, Gabriola Island, BC, Canada, 2000.

[RD05] Johanna Rothman and Esther Derby. *Behind Closed Doors: Secrets of Great Management*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2005.

[RG05] Jared Richardson and Will Gwaltney. *Ship It! A Practical Guide to Successful Software Projects*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2005.

[Rot98] Johanna Rothman. Defining and managing project focus. *American Programmer*, 11(2):19–23, 1998. http://www.jrothman.com/Papers/aparticle.html.

[Rot99] Johanna Rothman. How to use inch-pebbles when you think you can't. *American Programmer*, 12(5):24–29, 1999. http://www.jrothman.com/Papers/Howinch-pebbles.html.

[Rot02a] Johanna Rothman. Release criteria: Is this software done? STQE, 4(2):30–35, 2002.

[Rot02b] Johanna Rothman. What does success look like? *Stickyminds. com*, 2002. http://www.stickyminds.com/se/S3181.asp.

[Rot04a] Johanna Rothman. Got good rhythm? *Software Development Magazine*, 12(6), 2004. http://www.drdobbs.com/dept/architect/184415151.

[Rot04b] Johanna Rothman. *Hiring the Best Knowledge Workers, Techies, and Nerds: The Secrets and Science of Hiring Technical People*. Dorset House, New York, 2004.

[Roy70] Winston W. Royce. Managing the development of large software systems. *Proceedings, IEEE WECON*, pages 1–9, August 1970.

[Sch04] Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, Redmond, WA, 2004.

[SH06a] Venkat Subramaniam and Andy Hunt. *Practices of an Agile Developer*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2006.

[SH06b] Venkat Subramaniam and Andy Hunt. *Practices of an Agile Developer: Working in the Real World*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2006.

[SR98] Preston G. Smith and Donald G. Reinertson. *Developing Products in Half the Time: New Rules, New Tools, second ed*. John Wiley & Sons, New York, NY, 1998.

[TCKO00] Stephanie Teasley, Lisa Covi, M. S. Krishnan, and Judith S. Olson. How does radical collocation help a team succeed? *Proceedings of CSCW'00*, pages 339–346, 2000.

[Wei85] Gerald M. Weinberg. *The Secrets of Consulting*. Dorset House, New York, 1985.

[Wei92] Gerald M. Weinberg. *Quality Software Management: Volume 1, Systems Thinking*. Dorset House Publishing, Inc., New York, 1992.

[Wei94] Gerald M. Weinberg. *Quality Software Management, Volume 3: Congruent Action*. Dorset House, New York, 1994.

[Wei97] Gerald M. Weinberg. *Quality Software Management: Volume 4, Anticipating Change*. Dorset House Publishing, Inc., New York, 1997.

[Wie00] Karl Wiegers. Stop promising miracles. *Software Development*, 8(2), 2000.

[Wie05] Karl Wiegers. *Project Initiation Handbook*. Process Impact, Clackamas, OR, 2005.

[WJ77] Bruce W.Tuckman and Mary Ann C. Jensen. Stages of small group development revisited. *Group and Organizational Studies*, 2:419– 427, 1977.

[WJC00] Robert Wysocki, Robert Beck Jr., and David B. Crane. *Effective Project Management, second edition*. John Wiley & Sons, New York, NY, 2000.

[WK02] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley, Reading, MA, 2002.

[You99] Edward Yourdon. *Death March: The Complete Software Developer's Guide to Surviving âĂŸMission Impossible' Projects*. Prentice Hall, Englewood Cliffs, NJ, 1999.

"她从多年的一线工作经验中萃取出精华，以深入浅出的方式展现给读者。……就我来说，即使已经有了这么多年的项目和工程管理经验，还是可以在书中发现新东西。当我面临全新挑战，需要一块坚固的甲板帮我渡过激流险滩时，Johanna一定就是我要找的那个人。"

——艾伦·R.索尔兹伯里，Yahoo伯克利研究中心前总监

"作为一个软件工程及项目管理的老兵，我认为这是项目管理领域中最佳的实践书，尤其是作者在项目管理上积累了多年的丰富经验，肯定能帮项目经理解决相当多的疑难杂症。……好东西要分享，在此我诚挚地向大家推荐这本好书，分享我在其中得到的共鸣与喜悦。"

——胡百师，微软项目管理MVP

Manage It! Your Guide to Modern, Pragmatic Project Management

项目管理修炼之道

项目管理对于项目成败至关重要，项目经理往往面临着巨大的压力和挑战：虽然已经有很多项目管理理论和方法，但实践中每个项目都有自己的独特性，没有现成的解决方案可以套用。

怎么办？这部荣获软件业奥斯卡——Jolt奖的著作给出了很好的解答。作者多年来帮助许多高科技公司成功地解决了各种有关产品开发管理的棘手问题，本书正是她宝贵实战经验的提炼。书中从应对实际风险的角度出发，讲述了从项目启动、项目规划到项目结束的整个管理流程；展示了作者的思考过程，从评估项目背景，选择生命周期，直到为项目建立清晰的条件；同时穿插了丰富的提示和真实案例，介绍了如何识别和解决日程安排中的常见问题。这些真知灼见不仅适用于软件项目管理，同样适用于其他产品的开发项目。

这是一本可使项目经理即刻上手的名副其实的实战指南。任何类型的项目经理，无论你是使用瀑布式、迭代式还是敏捷式生命周期模型，都应该反复研读本书，从中得到有益的提示和帮助。

- 理论让路，实践先行，Amazon五星畅销著作
- 项目经理的必备实战手册
- 众多实例，犹如真实场景再现

The Pragmatic Programmers

本书相关信息请访问：图灵网站 http://www.turingbook.com
读者/作者热线：(010)51095186
反馈/投稿/推荐信箱：contact@turingbook.com

分类建议 计算机/软件工程/项目管理

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-21361-7
定价：49.00元